6. SDP Toolkit Specification

6.1 Introduction

In this section, we give a descriptive list of Toolkit software tools designed to satisfy the requirements listed in *PGS Toolkit Requirements Specification for the ECS Project*, Hughes Information Technology Systems, Inc. 193-801-SD4-001, October 1993 and updated in July 1995. The following fields are provided: a name, a synopsis field, a description of each tool, a list of input and output, an error return field, examples, notes, and a cross reference to the target Toolkit requirement(s).

It is assumed that ECS science software requests for system services, for system and resource accesses, file I/O requests, error message transaction, metadata formatting, accesses to spacecraft orbit and attitude, and time and date requests must be made through the Toolkit, as explained in section 4.1. This usage will be tested at integration time at the DAACs. These tools are described in Section 6.2. Other services, such as geographic information data base requests, geolocation tools, scientific and math library calls, requests for physical constants and unit conversions, will be provided; their usage will be encouraged, but not enforced. They are the subject of Section 6.3.

Toolkit routines use the following naming convention:

PGS_GROUPNAME_FUNCTIONALNAME. The GROUPNAME denotes the function of that group of Toolkit routines: IO=Input/Output, SMF=Status/message Facility, MEM=Memory Management, MET=metadata, EPH=Ephemeris/Attitude data access, TD=time and date conversion, PC=ProcessControl, DEM=Digital Elevation Model access, AA=Ancillary Data Access, CBP=Celestial Body Position, GCT=Geo-coordinate Transformation, CUC=Constant and Unit Conversion, CSC=Coordinate System Conversion. The remaining part of the name has sufficient detail to indicate the functionality of the tool. (See also Section 3.2)

There are several C (.h) and FORTRAN (.f) include files listed in the tool descriptions in the following sections, e.g., PGS_IO.h. These files are meant to contain descriptions of data structures, constants; headers; configuration information for data files called by the tools; common symbols; return codes, etc., used in that section. To view these files, look in Toolkit directory $PGSHOME/include.

A note on error handling: Since each function has only one return value; every effort has been made to preserve the most important warning or error value on returning. Given that subordinate functions often have several possible returns, and different users have different priorities, it is always advisable to check the message log in $PGSRUN as well as examining the return. When totally inconsistent behavior is found in a return from a subordinate function, the returned value is PGS_E_TOOLKIT. Example: a Toolkit function passes an internally generated vector, whose length is certain to be nonzero~~,~~ to a subordinate function. The lower-level function then returns a warning or error return saying that the vector is of zero length; while the higher level function returns PGS_E_TOOLKIT. Another example: if a valid spacecraft tag is passed in, but rejected as invalid down the processing line, the error PGS_E_TOOLKIT is returned by the higher-level function. Thus return value PGS_E_TOOLKIT indicates a flaw in the software, the violation of an array boundary, a hardware, compiler, or system error, corrupted data, or some similarly serious condition that invalidates the processing.

6.2 SDP Toolkit Tools-Mandatory

6.2.1 File I/O Tools

This section describes the set of tools used to perform file I/O, including Level 0 access generic and temporary I/O tools, also proposed metadata tools. An explanation of usage of the Toolkit as regards Hierarchical Data Format (HDF) is also included.

6.2.1.1 Level 0 Science Data Access Tools

6.2.1.1.1 Introduction

These Level 0 access tools are used to open and read data from Level 0 data files. These files are generated and formatted by EDOS for AM and PM platform data, and by the science data processing facility (SDPF) for TRMM platform data. The Level 0 access tools also support ADEOS-II Level 0 data files.

The Level 0 access tool design has simple user interfaces, and allows science software to do much of the data unpacking in whatever manner is desired. Essentially all header and packet data are returned in character buffers. The packet data is returned a single packet at a time, so the science software can decide whether to store it or to immediately process it.

This delivery of L0 tools is preliminary in anticipation of receipt of definitive EDOS file header formats. TRMM and EOS AM L0 data formats have been implemented to the extent possible; however, little is known about EOS PM and ADEOS-II L0 file formats other than the general form of the packet data-file header format is undefined at this writing. We await receipt of L0 file format definitions from ADEOS-II and from EDOS for EOS PM. In addition no attempt was made in the current version of the prototype to optimize speed of the L0 processing tools.

A complete specification of the Level 0 file formats used in construction of this software is found in Appendix F.

6.2.1.1.2 Design Overview

The design focuses on the idea of a “virtual data set, consisting of all staged physical L0 files for a particular data type. By data type is meant data that are related in some way; most often this means data with a common application process identifier (APID). There may be many virtual data sets for a given production run. For example, main Clouds and Earth Radiant Energy System (CERES) L0 processing involves science data (APID 54) and housekeeping data (all other APIDs). Each of these two sets of data corresponds to a single virtual data set in the

Level 0 tool design. Each virtual data set corresponds to a single logical file ID in the science software and (at the SCF) in the Process Control File (PCF).

For a given run, if a given set of data for a single set of data (science or housekeeping) needs to be broken into more than one file, then each physical file corresponds to a different version of the same logical file ID in the PCF. (This is never expected to be the case for TRMM, but may be for EOS AM or PM.)

Next is given a brief summary of the functions of the L0 tools. The tools are divided into two groups: one group consisting of required tools for reading L0 data in production software, and one group for use only at the SCF for generation of test data sets.

6.2.1.1.3 Tools for Reading Production L0 Data

PGS_IO_L0_Open sets up internal tables that allow the SDP Toolkit to provide the science software with time-ordered access to file attributes. It opens the first physical file and positions the file pointer at the earliest packet in the staged data. It returns the virtual file handle used by other L0 access tools.

PGS_IO_L0_SetStart is for optionally positioning the virtual file pointer at a start time that is different from the earliest packet in the staged data.

PGS_IO_L0_SetStartCntPkts is for optionally positioning the virtual file pointer at a start time that is different from the earliest packet in the staged data. Also tracks the number of packets skipped in the current file

PGS_IO_L0_GetHeader is for retrieving data from the physical L0 file header; in addition, for TRMM processing, it retrieves data from the file footer, which consists of quality and missing packet information. Data is returned in a simple character buffer.

PGS_IO_L0_GetPacket retrieves a single packet’s worth of data. Data is also returned in a simple character buffer by this function.

PGS_IO_L0_Close is for closing a L0 virtual data set.

6.2.1.1.4 Tools for Generating Simple Simulated L0 Data Sets

The above tools satisfy SDP Toolkit requirements for tools that read Level 0 data files; along with these, a means is provided to generate simple simulated Level 0 files. A major portion of TRMM Level 0 processing may be simulated using these files; for EOS AM and PM platforms, lack of file format definition has prevented more than the packet simulation included in the simulator. EOS AM users can employ the TRMM header formats temporarily.

Provided for simulated file generation are:

L0sim, an executable interactive utility that queries the user about parameters used in creation of a simulated Level 0 data set. It can create file(s) for a single APID, or a housekeeping file with many APIDs; one or many physical files per APID; and many other things. See Appendix E for an example of its use.

PGS_IO_L0_File_Sim, a function callable from C or FORTRAN; it is the underlying function used by *L0sim*. Users who prefer to customize file simulations to fit their own needs may use this function.

6.2.1.1.5 Use Of L0 Read Tools In Science Software Processing

Next is presented a brief summary of how science software might use the L0 read tools to do Level 0 processing. A full example of L0 processing using CERES as an example is given in Appendix E. Examples are also provided in individual tool descriptions below.

In the production system, once the required L0 data and other data are staged, the PGE kicks off automatically. During development at the SCF, the developer must first generate file(s) using the simulator tools, then prepare entries in the Process Control File (PCF).

The science code might proceed as follows:

a. Call PGS_IO_L0_Open; with the logical file ID as input parameter used in the PCF. Get back a virtual file handle for use in other tools.

b. Optionally call PGS_PC_GetFileAttr or PGS_PC_GetFileByAttr to read an “attribute” file associated with the L0 data file. For example, for TRMM this might be the detached standard formatted data unit (SFDU) header file.

c. Optionally call PGS_PC_SetStart if a starting time other than the earliest in the data set is desired.

d. Allocate memory for as much data as is desired to save, based on the start and stop times returned from PGS_IO_L0_Open. (In FORTRAN 77 this will have to be hardcoded to some maximum.)

e. While there is still data left, first call PGS_IO_L0_GetHeader to read the physical file header, and also the footer (TRMM quality and accounting capsule (QAC) and missing data unit list (MDUL) data).

f. Call PGS_IO_L0_GetPacket to read a single packet. Repeat until end of data reached, storing the data as desired.

g. If PGS_IO_L0_GetPacket returns a value indicating a new physical file has been opened, loop back to call PGS_IO_L0_GetHeader again to read the new file header.

h. Call PGS_IO_L0_Close to close this virtual data set.

i. If there are more virtual data sets (e.g., APIDs) to process, loop back to call PGS_IO_Gen_Open again.

Note that this algorithm is just one example of how this might be done. Another way is to open several virtual data sets at once.

Please note also that science software is responsible for unpacking headers, packets and footers as it sees fit. Specification of their formats as used in this version of the software appears in Appendix F.

6.2.1.1.6 Open Issues

A major limitation in designing these tools was and is lack of ADEOS-II and EOS PM file format definition, other than the packet format. We await this information from ADEOS-II and EDOS respectively.

Most aspects of the TRMM file are handled by the read tools and the simulator. One item that is not implemented in this prototype is the internal structure of the quality data and missing data list. This means that if the user wants to simulate quality data or missing data, s/he will have to construct it him/herself, then use the function PGS_IO_L0_File_Sim to generate files. In addition s/he will have to write code to make use of such data.

In this Toolkit delivery, no attempt has been made to optimize for speed. (This applies to the tool PGS_IO_L0_GetPacket.)

Please note that this code as delivered is preliminary until definitive file header formats are received from ADEOS-II, EDOS and Pacor/DDF.

Feedback from the science teams concerning design and implementation of the prototype is strongly encouraged.

6.2.1.1.7 Special Note on Processing TRMM and ADEOS-II Files

In order to process the Level 0 data files the Level 0 access tools must be able to convert the time found in the data files to TAI. Special preparation is required to do this in the case of TRMM and ADEOS-II.

To properly convert times to or from TRMM s/c clock time the value of the TRMM Universal Time Correlation Factor (UTCF) must be known. This value must be supplied by the user in the Process Control File (PCF). The following line MUST be contained in the PCF for any process that is converting to or from TRMM s/c clock time:

10123|TRMM UTCF value|<UTCF VALUE>

Where the proper value of the UTCF should be substituted for <UTCF VALUE>.

To properly convert times to or from ADEOS-II s/c clock time the ADEOS-II Time Differential (TMDF) values must be known. These values must be supplied by the user in the Process Control File (PCF). The following lines MUST be contained in the PCF for any process that is converting to or from ADEOS-II s/c clock time:

<UTC VALUE>

10120|ADEOS-II s/c reference time|<S/C REFERENCE TIME>
10121|ADEOS-II ground reference time|<GROUND REFERENCE TIME>
10122|ADEOS-II s/c clock period|<S/C PERIOD>

Where:

the proper value of the S/C clock reference time should be substituted for
< S/C REFERENCE TIME>.

the proper value of the ground reference time should be substituted for <GROUND REFERENCE TIME> (this time should be in TAI format-see sec. 6.2.7 Time and Date Conversion Tools).

the proper value of the S/C clock period should be substituted for <S/C PERIOD>.

Open a Virtual Data Set

NAME: **PGS_IO_L0_Open**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_Open(
          PGSt_PC_Logical                  file_logical,
          PGSt_tag                         spacecraft_tag,
          PGSt_IO_L0_VirtualDataSet        *virtual_file,
          PGSt_double                      *start_time,
          PGSt_double                      *stop_time)
```

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_TD.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'

integer function
PGS_IO_L0_Open(
+         file_logical,
+         spacecraft_tag,
+         virtual_file,
+         start_time,
+         stop_time)

          integer file_logical
          integer spacecraft_tag
          integer virtual_file
          double precision start_time
          double precision stop_time
```

DESCRIPTION This tool opens the virtual data set pointed to by file_logical. A virtual Level 0 data set is defined by the set of physical data files that have been staged for this Level 0 process.

The tool returns a descriptor that is used by all the Level 0 tools to access the specified virtual data set. The tool also returns the start and stop times of this virtual data set.

INPUTS: file_logical-The logical file descriptor for this virtual data set, as given in the Process Control File

spacecraft_tag-The tag identifying which of the supported spacecraft platforms generated this virtual data set. Must be either PGSd_TRMM, PGSd_EOS_AM, PGSd_EOS_PM or PGSd_ADEOS_II.

OUTPUTS: virtual_file-The file descriptor used by all other Level 0 access tools to refer to the virtual data set

start_time-The start time of this virtual data set

stop_time-The stop time of this virtual data set

Time format is TAI: continuous seconds since 12AM UTC Jan. 1, 1993

RETURNS:

Table 6-1. PGS_IO_L0_Open Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_W_L0_CORRUPT_FILE_HDR	Corrupted file header
PGSIO_E_L0_BAD_SPACECRAFT_TAG	Invalid spacecraft tag
PGSIO_E_L0_INIT_FILE_TABLE	Error during read of physical file header for initialization
PGSIO_E_L0_INVALID_FILE_LOGICAL	Failed to process this file logical in process control file
PGSIO_E_L0_MAP_VERSIONS	Failed to initialize internal physical file table
PGSIO_E_L0_PHYSICAL_OPEN	Unable to open physical file
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_E_L0_SEEK_1ST_PACKET	Can’t find 1st packet in dataset

EXAMPLES: Prepare in part for Lightening Imaging Sensor (LIS) Level 0 processing by opening the LIS/TRMM Level 0 virtual data set for science APID 61.

For TRMM, there is expected to be only one physical file per APID per day. In this case each virtual data set (APID) corresponds to exactly one physical file.

At the SCF, you must prepare entries of the following form in the Process Control File:

```
? PRODUCT INPUT FILES
# [ set env var PGS_PRODUCT_INPUT for default location ]
#
61|TRMM_G0091_1997-11-
    01T00:00:00Z_dataset_V01_01||||TRMM_G0091_1997-11-
    01T00:00:00Z_sfdu_V01_01|1
```

(Here the logical ID used is arbitrarily set to the APID.)

Note: In the above Process Control File entry, the file name in the next-to-last field is the TRMM SFDU header file, which is a file that contains data associated with the given L0 file. Use functions PGS_IO_PC_GetFileAttr or PGS_IO_PC_GetFileByAttr to retrieve data from this file. Also, the PCF entry must appear on a single line, and not be broken into several lines as shown here.

C:

```
#define SCIENCE_FILE 61

PGSt_IO_L0_VirtualDataSet     virtual_file;
PGSt_PC_Logical               file_logical;
PGSt_tag                      spacecraft_tag;
PGSt_double                   start_time;
PGSt_double                   stop_time;
PGSt_SMF_status               returnStatus;

file_logical = SCIENCE_FILE;
spacecraft_tag = PGSd_TRMM;

returnStatus = PGS_IO_L0_Open(
      file_logical,
      spacecraft_tag,
      &virtual_file,
      &start_time,
      &stop_time);

/#      Virtual file handle virtual_file may now be used as
input to other L0 access tools #/
```

FORTRAN:

```
implicit none

INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_TD.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'
integer     SCIENCE_FILE

parameter (SCIENCE_FILE=61)
integer           pgs_io_l0_open
integer           file_logical
integer           spacecraft_tag
integer           virtual_file
double precision  start_time
```

```
                        double precision  stop_time
                        integer           returnstatus

                        file_logical = SCIENCE_FILE
                        spacecraft_tag = PGSd_TRMM

                        returnstatus = pgs_io_l0_open(
                              file_logical,
                              spacecraft_tag,
                              virtual_file,
                              start_time,
                              stop_time)
C                       Virtual file handle virtual_file may now be used as input to
C                       other L0 access tools
```

NOTES: A virtual data set is defined by a set of one or more related Level 0 physical files. For example, it might consist of all physical files corresponding to a single TRMM science application ID (APID) for a single production run. In the case of EDOS formatted Level 0 data files, a virtual data set consists of all physical files comprising an EDOS PDS/EDS. Only one PDS/EDS is allowed per virtual file.

The maximum number of virtual data sets that may be open at any one time is 20.

This function must be called first; before any other Toolkit Level 0 access tools are called.

A virtual data set may consist of several physical files. In this case the files are listed in the process control file with the same logical ID (1st field) but different instance number (last field).

The physical file version corresponding to the first time-ordered set of packets for the virtual data set is opened by this tool. The file pointer is left positioned so that the next call to PGS_IO_L0_GetPacket will read the first packet in the file.

To get file header and footer (TRMM only) information for the newly opened physical file, use tool PGS_IO_L0_GetHeader. A rudimentary check is done on the header of the first physical file of the virtual data set. If an error is found in the header this function will return the value PGSIO_W_L0_CORRUPT_HEADER. The file will be opened anyway and the user may use the function PGS_IO_L0_GetHeader() to retrieve the header. That function will give a more detailed analysis of the problem. Users should be aware, though, that if they proceed after getting the return PGSIO_W_L0_CORRUPT_HEADER from this function they do so at THEIR OWN RISK. This return value indicates that the file header is corrupt and the use of any further Toolkit functions to attempt to read the file may produce unexpected results.

In the case of EDOS formatted Level 0 data files (PDS/EDS) the “header” returned will actually be the Construction Record.

RELEASE NOTES:

This function conforms to EDOS-EGS ICD (June 28, 1996)

See Section 6.2.1.1.6 Open Issues

Note Regarding Use of the Process Control File:

If more than one physical file is associated with a given virtual data set, the entries in the Process Control File that map the data set from file_logical to the physical files must appear in reverse numerical order. For example, in a three-file data set, file instance #3 is listed first and file instance #1 is listed last. This mechanism will become transparent in the production system.

REQUIREMENTS: PGSTK-0140, PGSTK-0190, PGSTK-0240

Set Start Time

NAME: **PGS_IO_L0_SetStart**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_SetStart(
        PGSt_IO_L0_VirtualDataSet       virtual_file,
        PGSt_double                     start_time)
```

FORTRAN:

```
INCLUDE    'PGS_SMF.f'
INCLUDE    'PGS_PC.f'
INCLUDE    'PGS_PC_9.f'
INCLUDE    'PGS_TD.f'
INCLUDE    'PGS_IO.f'
INCLUDE    'PGS_IO_1.f'

integer function PGS_IO_L0_SetStart(virtual_file, start_time)
        integer              virtual_file
        double precision     start_time
```

DESCRIPTION Sets the virtual file pointer so that the next call to the tool PGS_IO_L0_GetPacket will read the first available packet at or after the specified time.

INPUTS: virtual_file-The file descriptor for this virtual data set, returned by the call to PGS_IO_L0_Open

start_time-The start time of the desired packet. Format is TAI: continuous seconds since 12AM UTC Jan. 1, 1993.

OUTPUTS: NONE

RETURNS:

Table 6-2. PGS_IO_L0_SetStart Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_VIRTUAL_DS_NOT_OPEN	Virtual data set is not open
PGSIO_W_L0_TIME_NOT_FOUND	Requested start time not found; file pointer position was unchanged
PGSIO_W_L0_PHYSICAL_CLOSE	Failed to close physical file
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_E_L0_PHYSICAL_OPEN	Unable to open physical file
PGSIO_E_L0_SEEK_PACKET	Unable to find requested packet
PGSIO_M_L0_HEADER_CHANGED	New physical file open-file header has changed

EXAMPLES: Set the time to start processing at 20 minutes after the data set start time. Examples assume the data set start time has previously been returned from PGS_IO_L0_Open.

C:

```
PGSt_IO_L0_VirtualDataSet     virtual_file;
PGSt_double                   start_time;
PGSt_double                   new_start_time;
PGSt_SMF_status               returnStatus;

new_start_time = start_time + 1200.0;

returnStatus = PGS_IO_L0_SetStart( virtual_file,
      new_start_time);
if (returnStatus != PGS_S_SUCCESS)
{
    goto EXCEPTION;  /# GO TO EXCEPTION HANDLING #/
}
```

FORTRAN:

```
implicit none

INCLUDE     ‘PGS_SMF.f’
INCLUDE     ‘PGS_PC.f’
INCLUDE     ‘PGS_PC_9.f’
INCLUDE     ‘PGS_TD.f’
INCLUDE     ‘PGS_IO.f’
INCLUDE     ‘PGS_IO_1.f’

integer           pgs_io_l0_setstart
integer           virtual_file
double precision  start_time
double precision  new_start_time
integer           returnstatus

new_start_time = start_time + 1200.0

returnstatus = pgs_io_l0_setstart( virtual_file,
      new_start_time)
```

```
if (returnStatus .ne. PGS_S_SUCCESS) goto EXCEPTION
```

NOTES: Normal return is PGS_S_SUCCESS.

A virtual data set must have been opened by PGS_IO_L0_Open before this function is called.

RELEASE NOTES:

See Section 6.2.1.1.6 Open Issues

REQUIREMENTS: PGSTK-0140, PGSTK-0200, PGSTK-0220, PGSTK-0240

Set Start Time and Count Packets

NAME: **PGS_IO_L0_SetStartCntPkts**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_SetStart(
          PGSt_IO_L0_VirtualDataSet          virtual_file,
          PGSt_double                        start_time
          PGSt_integer*                      totpacket_skip)
```

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_TD.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'

integer function PGS_IO_L0_SetStart(virtual_file, start_time,totpacket_skip)
          integer               virtual_file
          double precision      start_time
          integer               totpacket_skip
```

DESCRIPTION Sets the virtual file pointer so that the next call to the tool PGS_IO_L0_GetPacket will read the first available packet at or after the specified time. Also tracks the number of packets skipped in the current file.

INPUTS: virtual_file-The file descriptor for this virtual data set, returned by the call to PGS_IO_L0_Open

start_time-The start time of the desired packet. Format is TAI: continuous seconds since 12AM UTC Jan. 1, 1993.

OUTPUTS: totpacket_skip - The total number of packets skipped before the desired packet selected at the specified time

RETURNS:

Table 6-3. PGS_IO_L0_SetStart Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_VIRTUAL_DS_NOT_OPEN	Virtual data set is not open
PGSIO_W_L0_TIME_NOT_FOUND	Requested start time not found; file pointer position was unchanged
PGSIO_W_L0_PHYSICAL_CLOSE	Failed to close physical file
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_E_L0_PHYSICAL_OPEN	Unable to open physical file
PGSIO_E_L0_SEEK_PACKET	Unable to find requested packet
PGSIO_M_L0_HEADER_CHANGED	New physical file open-file header has changed

EXAMPLES: Set the time to start processing at 20 minutes after the data set start time. Examples assume the data set start time has previously been returned from PGS_IO_L0_Open.

C:

```
PGSt_IO_L0_VirtualDataSet     virtual_file;
PGSt_double                   start_time;
PGSt_double                   new_start_time;
PGSt_SMF_status               returnStatus;
PGSt_integer            totalpacket_skip;

new_start_time = start_time + 1200.0;

returnStatus = PGS_IO_L0_SetStart( virtual_file,
      new_start_time, &totalpacket_skip);
if (returnStatus != PGS_S_SUCCESS)
{
    goto EXCEPTION;  /# GO TO EXCEPTION HANDLING #/
}
```

FORTRAN:

```
implicit none

INCLUDE     ‘PGS_SMF.f’
INCLUDE     ‘PGS_PC.f’
INCLUDE     ‘PGS_PC_9.f’
INCLUDE     ‘PGS_TD.f’
INCLUDE     ‘PGS_IO.f’
INCLUDE     ‘PGS_IO_1.f’

integer           pgs_io_l0_setstart
integer           virtual_file
integer           totalpacket_skip
double precision  start_time
double precision  new_start_time
integer           returnstatus
```

```
new_start_time = start_time + 1200.0

returnstatus = pgs_io_l0_setstart( virtual_file,
      new_start_time,totalpacket_skip)
if (returnStatus .ne. PGS_S_SUCCESS) goto EXCEPTION
```

NOTES: Normal return is PGS_S_SUCCESS.

A virtual data set must have been opened by PGS_IO_L0_Open before this function is called.

RELEASE NOTES:

See Section 6.2.1.1.6 Open Issues

REQUIREMENTS: PGSTK-0140, PGSTK-0200, PGSTK-0220, PGSTK-0240

Get Header Data

NAME: **PGS_IO_L0_GetHeader**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_GetHeader(
        PGSt_IO_L0_VirtualDataSet virtual_file,
        PGSt_integer                        header_buffer_size,
        PGSt_IO_L0_Header                   *header_buffer,
        PGSt_integer                        footer_buffer_size,
        PGSt_IO_L0_Footer                   *footer_buffer)
```

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_TD.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'

integer function PGS_IO_L0_GetHeader(virtual_file, header_buffer_size,
                                     header_buffer,
                                     footer_buffer_size,
                                     footer_buffer)
        integer virtual_file
        integer header_buffer_size
        character*(*)   header_buffer
        integer footer_buffer_size
        character*(*)   footer_buffer
```

DESCRIPTION: This tool reads header and footer information for the currently open physical file into the user-supplied buffers. It is intended to be called whenever the file header and footer data change, though it may be called at any time. In the case EDOS formatted files this tool will return the entire contents of the PDS/EDS Construction Record.

The file header and footer data will change whenever a call to one of the tools causes a new physical file to be opened. This will always occur upon a call to PGS_IO_L0_Open, and may also occur upon calls to PGS_IO_L0_SetStart and PGS_IO_L0_GetPacket. These latter two signal this event via a return status code of PGSIO_M_L0_HEADER_CHANGED. In the case of EDOS files, which have no headers, no notice will be given when a new physical file is opened. Typical use of this tool is in a loop of calls to read data packets.

INPUTS: virtual_file-The file descriptor for this virtual data set, returned by the call to PGS_IO_L0_Open

header_buffer_size-Size in bytes of user-supplied header buffer

footer_buffer_size-Size in bytes of user-supplied footer data buffer. If 0, do not read footer data (TRMM only)

OUTPUTS: header_buffer-User-supplied buffer containing the header, read in from the current physical file

footer_buffer-User-supplied buffer containing the footer data, read in from the current physical file (TRMM only)

RETURNS:

Table 6-4. PGS_IO_L0_GetHeader Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_BAD_BUF_SIZ	Buffer size must be a positive integer
PGSIO_E_L0_VIRTUAL_DS_NOT_OPEN	Virtual data set is not open
PGSIO_E_L0_FSEEK	Failed to locate requested byte in file
PGSIO_W_L0_HDR_TIME_ORDER	Time of last packet is earlier than first packet in file header
PGSIO_E_L0_BAD_VAR_HDR_SIZE	Size of the variable header is invalid
PGSIO_W_L0_BAD_PKT_DATA_SIZE	Total size of packet data is invalid
PGSIO_W_L0_BAD_PACKET_COUNT	Total number of packets is invalid
PGSIO_W_L0_BAD_FOOTER_SIZE	Size of the file footer is invalid
PGSIO_W_L0_ZERO_PACKET_COUNT	Total number of packets is zero
PGSIO_W_L0_HDR_BUF_TRUNCATE	Insufficient header buffer size - data
PGSIO_W_L0_FTR_BUF_TRUNCATE	Insufficient footer buffer size - data
PGSIO_W_L0_ALL_BUF_TRUNCATE	Insufficient header buffer AND footer buffer sizes - data truncated
PGSIO_E_L0_UNEXPECTED_EOF	Encountered unexpected end-of-file
PGS_E_UNIX	UNIX error (check log file for type of error)
PGSIO_E_L0_BAD_SPACECRAFT_TAG	Invalid spacecraft tag

EXAMPLES: The example shows how to use this function in conjunction with PGS_IO_L0_GetPacket to read Level 0 data from a single virtual data set. This algorithm works whether the virtual data set consists of only one, or of several physical files. All data in the virtual data set are read.

For clarity, error handling is omitted from the examples.

C:

```
#define HEADER_BUFFER_MAX    556  /# max # header bytes #/
#define FOOTER_BUFFER_MAX 100000  /# max # footer bytes #/
#define PACKET_BUFFER_MAX   7132  /# max # packet bytes #/

PGSt_IO_L0_VirtualDataSet  virtual_file;

PGSt_IO_L0_Header       header_buffer[HEADER_BUFFER_MAX];
PGSt_IO_L0_Footer       footer_buffer[FOOTER_BUFFER_MAX];
PGSt_IO_L0_Packet       packet_buf[PACKET_BUFFER_MAX];

PGSt_integer file_loop_flag;
PGSt_integer packet_loop_flag;

file_loop_flag = 1;
while( file_loop_flag )
{
   returnStatus = PGS_IO_L0_GetHeader( virtual_file,
                    HEADER_BUFFER_MAX, header_buffer,
                    FOOTER_BUFFER_MAX, footer_buffer );

/#   Unpack and/or save or process header and footer data
      here #/

   packet_loop_flag = 1;
   while( packet_loop_flag )
   {
      returnStatus = PGS_IO_L0_GetPacket(
                   virtual_file, PACKET_BUFFER_MAX,
                   packet_buf );

      switch (returnStatus)
      {
         case PGSIO_M_L0_HEADER_CHANGED:
         /# end of this physical file #/
            packet_loop_flag = 0;
            break;

         case PGSIO_W_L0_END_OF_VIRTUAL_DS:
        /# end of this virtual data set #/
           file_loop_flag = 0;
           packet_loop_flag = 0;
           break;
       }

/#   Unpack and/or save or process packet data here #/

   }     /# End while (packet_Loop_flag) #/

}    /# End while (file_Loop_flag) #/
```

FORTRAN:

```
implicit none

INCLUDE     ‘PGS_SMF.f’
INCLUDE     ‘PGS_PC.f’
INCLUDE     ‘PGS_PC_9.f’
INCLUDE     ‘PGS_TD.f’
INCLUDE     ‘PGS_IO.f’
INCLUDE     ‘PGS_IO_1.f’

character*556     header_buffer
character*7132    packet_buffer
character*100000  footer_buffer
integer           pgs_io_l0_getheader
integer           pgs_io_l0_getpacket
integer           virtual_file
integer           file_loop_flag
integer           packet_loop_flag
integer     returnstatus

file_loop_flag = 1
do while( file_loop_flag )

   returnstatus = pgs_io_l0_getheader( virtual_file,
         556, header_buffer,
         100000, footer_buffer )

C   Unpack and/or save or process header and footer data here

packet_loop_flag = 1

do while( packet_loop_flag )

returnStatus = pgs_io_l0_getpacket(

virtual_file, PACKET_BUFFER_MAX, packet_buf )

if (returnstatus .eq. PGSIO_M_L0_HEADER_CHANGED) then

C   end of this physical file

packet_loop_flag = 0

                           else if (returnstatus .eq.
                              PGSIO_W_L0_END_OF_VIRTUAL_DS) then

C    end of this virtual data set

file_loop_flag = 0

packet_loop_flag = 0

end if
```

```
C   Unpack and/or save or process packet data here

end do

end do
```

NOTES: Memory must be allocated to the output buffers before this tool is called. Failure to do this may result in a core dump. (In FORTRAN 77, the buffer CHARACTER array length must be hardcoded.)

If the tool determines that the actual size of the file header or footer is larger than the user-supplied buffer size, the header or footer data is truncated to fit the user buffer. In this case, the return status will be PGSIO_W_L0_HDR_BUF_TRUNCATE (if header buffer too small), PGSIO_W_L0_FTR_BUF_TRUNCATE (if footer buffer too small), or .PGSIO_W_L0_ALL_BUF_TRUNCATE (if both buffers too small).

To retrieve the header and footer information from the first physical file in a virtual data set, this tool must be called after first having opened the virtual data set using the tool PGS_IO_L0_Open. To retrieve the header and footer information from subsequent physical files within a virtual data set, this tool should be called after the science software receives the return status PGSIO_M_L0_HEADER_CHANGED from the tool PGS_IO_L0_GetPacket.

A virtual data set must have been opened by PGS_IO_L0_Open before this function is called. If the header of the currently open physical file is found to be corrupted, this function will return a warning to that effect:

PGSIO_W_L0_HDR_TIME_ORDER
PGSIO_E_L0_BAD_VAR_HDR_SIZE
PGSIO_W_L0_BAD_PKT_DATA_SIZE
PGSIO_W_L0_BAD_PACKET_COUNT
PGSIO_W_L0_BAD_FOOTER_SIZE
PGSIO_W_L0_ZERO_PACKET_COUNT

The above returns indicate an error was found in the file header. The header buffer will be returned, although it MAY be truncated. Similarly the footer buffer (TRMM only) may be truncated or even missing if the corrupt header file indicated that the start of the footer buffer was at an offset (in the file) greater than the size of the physical file. The user is cautioned to check the returned buffer(s) carefully in these cases. Further, the user is cautioned that while the function PGS_IO_L0_GetPacket() may still be called, that function may produce unexpected results if the file header is corrupt.

RELEASE NOTES:

This function conforms to EDOS-EGS ICD (June 28, 1996)

See Section 6.2.1.1.6 Open Issues

REQUIREMENTS: PGSTK-0140, PGSTK-0210, PGSTK-0230, PGSTK-0240

Get a Single Packet

NAME: **PGS_IO_L0_GetPacket**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_GetPacket(
        PGSt_IO_L0_VirtualDataSet       virtual_file,
        PGSt_integer                    packet_buffer_size,
        PGSt_IO_L0_Packet               *packet_buffer)
```

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_TD.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'

integer function PGS_IO_L0_GetPacket(virtual_file, packet_buffer_size,
                                     packet_buffer)
        integer virtual_file
        integer packet_buffer_size
        character*(*)   packet_buffer
```

DESCRIPTION: Reads a single data packet from a Level 0 virtual data set into the user-supplied buffer.

INPUTS: virtual_file-The file descriptor for this virtual data set returned by PGS_IO_L0_Open.

packet_buffer_size-Size in bytes of user-supplied packet buffer.

OUTPUTS: packet_buffer-User-supplied buffer containing the data packet read in from the specified virtual data set.

RETURNS:

Table 6-5. PGS_IO_L0_GetPacket Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_E_L0_PHYSICAL_NOT_OPEN	No physical file currently open for this virtual data set
PGSIO_E_L0_PKT_BUF_OVERFLOW	Packet buffer too small; no data was read
PGSIO_E_L0_UNEXPECTED_EOF	Encountered unexpected end-of-file
PGSIO_W_L0_PKT_BUF_TRUNCATE	Insufficient buffer size-data truncated
PGSIO_W_L0_END_OF_VIRTUAL_DS	Reached end of the current data set
PGSIO_M_L0_HEADER_CHANGED	New physical file open-file header has changed
PGSIO_E_L0_NEXT_PHYSICAL	Error opening next physical file in sequence
PGSIO_E_L0_SEEK_1ST_PACKET	Can't find first packet in dataset
PGSIO_W_L0_BUFTRUNC_END_DS	Insufficient packet buffer size-reached end of the current data set
PGSIO_W_L0_BUFTRUNC_HDR_CHG	Insufficient packet buffer size-new physical file open-file header has changed
PGSIO_E_L0_BUFTRUNC_NXTFILE	Insufficient buffer size-error opening next physical file in sequence
PGS_E_UNIX	UNIX error (check StatusLog file)

EXAMPLES: The example shows how to use this function in conjunction with PGS_IO_L0_GetPacket to read Level 0 data from a single virtual data set. This algorithm works whether the virtual data set consists of only one, or of several physical files. All data in the virtual data set are read.

For clarity, error handling is omitted from the examples.

C:

```
#define HEADER_BUFFER_MAX    556  /# max # header bytes #/
#define FOOTER_BUFFER_MAX 100000  /# max # footer bytes #/
#define PACKET_BUFFER_MAX   7132  /# max # packet bytes #/

PGSt_IO_L0_VirtualDataSet  virtual_file;

PGSt_IO_L0_Header       header_buffer[HEADER_BUFFER_MAX];
PGSt_IO_L0_Footer       footer_buffer[FOOTER_BUFFER_MAX];
PGSt_IO_L0_Packet       packet_buf[PACKET_BUFFER_MAX];

PGSt_integer file_loop_flag;
PGSt_integer packet_loop_flag;

file_loop_flag = 1;
while( file_loop_flag )
{
   returnStatus = PGS_IO_L0_GetHeader( virtual_file,
                    HEADER_BUFFER_MAX, header_buffer,
                    FOOTER_BUFFER_MAX, footer_buffer );

/#   Unpack and/or save or process header and footer data
      here #/

   packet_loop_flag = 1;
   while( packet_loop_flag )
   {
      returnStatus = PGS_IO_L0_GetPacket(
                   virtual_file, PACKET_BUFFER_MAX,
                   packet_buf );

      switch (returnStatus)
      {
         case PGSIO_M_L0_HEADER_CHANGED:
         /# end of this physical file #/
            packet_loop_flag = 0;
            break;

         case PGSIO_W_L0_END_OF_VIRTUAL_DS:
        /# end of this virtual data set #/
           file_loop_flag = 0;
           packet_loop_flag = 0;
           break;
       }

/#   Unpack and/or save or process packet data here #/

   }     /# End while (packet_loop_flag) #/

}    /# End while (file_loop_flag) #/
```

FORTRAN:

```
implicit none

INCLUDE     ‘PGS_SMF.f’
INCLUDE     ‘PGS_PC.f’
INCLUDE     ‘PGS_PC_9.f’
INCLUDE     ‘PGS_TD.f’
INCLUDE     ‘PGS_IO.f’
INCLUDE     ‘PGS_IO_1.f’

character*556     header_buffer
character*7132    packet_buffer
character*100000  footer_buffer
integer           pgs_io_l0_getheader
integer           pgs_io_l0_getpacket
integer           virtual_file
integer           file_loop_flag
```

```
                  integer           packet_loop_flag
                  integer     returnstatus

                  file_loop_flag = 1
                  do while( file_loop_flag )

                     returnstatus = pgs_io_l0_getheader( virtual_file,
                           556, header_buffer,
                           100000, footer_buffer )

C   Unpack and/or save or process header and footer data here

packet_loop_flag = 1

do while( packet_loop_flag )

returnStatus = pgs_io_l0_getpacket(

virtual_file, PACKET_BUFFER_MAX, packet_buf )

if (returnstatus .eq. PGSIO_M_L0_HEADER_CHANGED) then

C   end of this physical file

packet_loop_flag = 0

                           else if (returnstatus .eq.
                              PGSIO_W_L0_END_OF_VIRTUAL_DS) then

C    end of this virtual data set

file_loop_flag = 0

packet_loop_flag = 0

end if

C   Unpack and/or save or process packet data here

end do

end do
```

NOTES: Memory must be allocated to the output buffer before this tool is called. Failure to do this may result in a core dump. (In FORTRAN 77, the buffer CHARACTER array length must be hardcoded.)

Normal return is PGS_S_SUCCESS. If getting the next packet requires that a new physical file be opened, the header and quality data will change. In this case, the return status is set to PGSIO_M_L0_HEADER_CHANGED. This allows the user to test the return status and get updated header and quality data using the tool

PGS_IO_L0_GetHeader, in the case where there is more than one physical file per virtual data set.

If the tool determines that the size of the packet is larger than the user buffer size, as specified by the parameter packet_size, it will truncate the packet to fit the user buffer. In this case, the return status will be PGSIO_W_L0_BUFFER_TRUNCATE.

Packet formats for TRMM, EOS AM, and EOS PM are supported.

A virtual data set must have been opened by PGS_IO_L0_Open before this function is called.

This function returns no data if the packet buffer size is less than 6 bytes (the primary packet header size). It returns a warning and a truncated buffer if the packet buffer size is more than 6 bytes but less than the actual packet length.

RELEASE NOTES:

See Section 6.2.1.1.6 Open Issues

REQUIREMENTS: PGSTK-0140, PGSTK-0200, 0240

Close a Virtual Data Set

NAME: **PGS_IO_L0_Close**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_Close(
        PGSt_IO_L0_VirtualDataSet virtual_file)
```

FORTRAN:

```
INCLUDE   'PGS_SMF.f'
INCLUDE   'PGS_PC.f'
INCLUDE   'PGS_PC_9.f'
INCLUDE   'PGS_TD.f'
INCLUDE   'PGS_IO.f'
INCLUDE   'PGS_IO_1.f'

integer function PGS_IO_L0_Close(virtual_file)
        integer virtual_file
```

DESCRIPTION: This tool closes a virtual data set opened by a call to the tool PGS_IO_L0_Open.

INPUTS: virtual_file-The file descriptor for this virtual data set, returned by the call to PGS_IO_L0_Open.

OUTPUTS: NONE

RETURNS:

Table 6-6. PGS_IO_L0_Close Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_VIRTUAL_DS_NOT_OPEN	Virtual data set is not open
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_W_L0_PHYSICAL_CLOSE	Failed to close physical file

EXAMPLES: Close a virtual data set opened with a call to PGS_IO_L0_Open. Go to exception handling if there was an error.

C:

```
PGSt_SMF_status returnStatus = PGS_S_SUCCESS;
PGSt_IO_L0_VirtualDataSet  virtual_file;

returnStatus = PGS_IO_L0_Close(virtual_file);
if (returnStatus != PGS_S_SUCCESS) goto EXCEPTION;
```

FORTRAN:

```
implicit none

INCLUDE     ‘PGS_SMF.f’
INCLUDE     ‘PGS_PC.f’
INCLUDE     ‘PGS_PC_9.f’
INCLUDE     ‘PGS_TD.f’
INCLUDE     ‘PGS_IO.f’
INCLUDE     ‘PGS_IO_1.f’
integer     pgs_io_l0_close
integer     returnstatus
integer     virtual_file

returnstatus = pgs_io_l0_close(virtual_file)
if (returnstatus != PGS_S_SUCCESS) goto 9999
```

NOTES: If a physical file is currently open, PGS_IO_Gen_Close is called to close it. Otherwise this step is skipped. In either case, the return will be PGS_S_SUCCESS.

REQUIREMENTS: PGSTK-0140, PGSTK-0190

Create a Simulated Level 0 Data File

NAME: **PGS_IO_L0_File_Sim**

SYNOPSIS:

C:

```
#include <PGS_IO.h>
#include <PGS_IO_L0.h>

PGSt_SMF_status
PGS_IO_L0_File_Sim(
        PGSt_tag            spacecraftTag,
        PGSt_integer        appID[],
        PGSt_integer        firstPacketNum
        char                startUTC[28],
        PGSt_integer        numValues,
        PGSt_double         timeInterval,
        PGSt_integer        dataLength[],
        PGSt_integer        otherFlags[2],
        char                *filename,
        void                *appData,
        PGSt_uinteger       qualMissLen[2])
        void                *qualData)
        void                *missData)
```

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_TD.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'

integer function pgs_io_l0_write_pkt( spacecrafttag, appid,firstpacketnum,
                                      startutc, numvalues,
                                      timeinterval, datalength,
                                      otherflags, filename,appdata,
                                      qualmisslen, qualdata,
                                      missdata )
        integer             spacecrafttag
        integer             appid(*)
        integer             firstpacketnum
        character*27        startutc
        integer             numvalues
        double precision    timeinterval
        integer             datalength(*)
```

integer	otherflags(2)
character*(*)	filename
(any)	appdata
integer	qualmisslen(2)
(any)	qualdata
(any)	missdata

DESCRIPTION: This tool creates file(s) containing simulated Level 0 data, each of which has a file header, packet data, and a file footer. For TRMM, a detached SFDU header file is also created for each Level 0 data file.

INPUTS: spacecraftTag-The spacecraft identifier desired for the output data.

appID-Array of application process identifiers (APIDs), one for each packet to be generated

firstPacketNum-Value of Packet Sequence Count to use for the initial packet

startUTC-The UTC time of the first packet. Formats supported:

a) YYYY-MM-DDThh:mm:ss.dddddd
b) YYYY-DDDThh:mm:ss.dddddd

numValues-The number of packets to generate

timeInterval-Time interval (in seconds) between packets

dataLength-Array of lengths, in bytes, of the Application Data for each packet. Does not include lengths of primary and secondary packet headers.

otherFlags-Array of length 2 with file header values

otherFlags[0]: bit-packed "Processing Options" byte TRMM values:

bit 3 on-Redundant Data Deleted
bit 6 on-Data Merging
bit 7 on-RS Decoding
bits 1,2,4,5,8-always off

For example, to simulate Redundant Data Deleted and RS Decoding, turn bits 3 and 7 on, which is decimal 68.

So set otherFlags[0]=68.

otherFlags[1]: "Data type Flags" byte TRMM values:

otherFlags[1]=1, Routine production data
otherFlags[1]=2, Quicklook data

(NOTE: These two fields are simply written to the appropriate place in the file header; no processing is done in this function based on their values.)

filename-The name of the file to be created containing the L0 packets.

appData-Optional user-defined input of the packet application data field. Does not include packet header data.

In C, if appData=NULL, a block of data of length equal to the largest value in array dataLength is filled with zeroes, for each packet.

(The remaining inputs are for TRMM file footer processing only. They are ignored for other platforms.)

qualMissLen-Array of length 2 with file footer section lengths qualMissLen[0]: quality (QAC) buffer length if qualMissLen[0]=0, no quality data are written to the file qualMissLen[1]: missing data (MDUL) buffer length if qualMissLen[1]=0 or qualMissLen[0]=0, no missing data are written to the file (QAC length and MDUL length are always written to the file)

qualData-Quality and Accounting Capsule (QAC) data In C, if qualData=NULL, a block of data of length qualMissLen[0] is filled with zeroes and written to the file. (In FORTRAN you pass a zero-filled array for this.)

missData-Missing Data Unit List (MDUL) data In C, if missData=NULL, a block of data of length qualMissLen[1] is filled with zeroes and written to the file. (In FORTRAN you pass a zero-filled array for this.)

OUTPUTS: NONE

RETURNS:

Table 6-7. PGS_IO_L0_File_Sim Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_BAD_NUM_PKTS	Illegal number of packets
PGSIO_E_L0_BAD_APP_ID	At least 1 packet had a bad Application ID
PGSIO_E_L0_BAD_FIRST_PKTNUM	Illegal first packet number
PGSTD_E_SC_TAG_UNKNOWN	spacecraft tag is unknown or not currently supported
PGSIO_E_L0_BAD_DATA_LENGTH	At least 1 packet had a bad data length
PGSIO_E_L0_BAD_NUM_APP_IDS	Illegal number of differing Application IDs
PGSTD_E_TIME_FMT_ERROR	Error in ASCII time string format (generic format: YYYY-MM-DDThh:mm:ss.ddddddZ)
PGSTD_E_TIME_VALUE_ERROR	Error in ASCII time string value (e.g., hours > 23)
PGS_E_TOOLKIT	Unspecified Toolkit error (check StatusLog file)
PGS_E_UNIX	UNIX error (check StatusLog file)
PGSMEM_E_MAXSIZE	Maximum memory size reached: %d in bytes
PGSIO_E_L0_PHYSICAL_OPEN	Unable to open physical file
PGSTD_E_DATE_OUT_OF_RANGE	the input time is outside the range of allowable values for the spacecraft clock

EXAMPLES: Generate a CERES L0 science telemetry file named TRMM_G0088_1997-12-01T00:00:00Z_V01.dataset_01, containing 3 packets of different lengths, starting at midnight Dec. 1, 1997 and spaced at 6.6 second intervals; also add QAC and MDUL data, filled with zeroes.

C:

```
                  #define N 3

                  PGSt_tag     spacecraftTag = TRMM;
                  PGSt_integer appID[N] = {54,54,54};
                  PGSt_integer firstPacketNum = 1;
                  char         *startUTC = “1997-12-01T00:00:00”;
                  PGSt_integer numValues = N;
                  PGSt_double  timeInterval = 6.6;
                  PGSt_integer dataLength[N];
                  PGSt_integer otherFlags[2];
                  char         *filename
                          = “TRMM_G0088_1997-12-01T00:00:00Z_V01.dataset_01”;
                  char         appData[9000];
                  PGSt_uinteger qualMissLen[2]={28,16};
                  char         *qualData=NULL;
                  char         *missData=NULL;

PGSt_SMF_status returnStatus;

                  otherFlags[0] = 68; /* Redundant Data Deleted & RS Decoding
                                      */
                  otherFlags[1] =  1; /* Routine production data */

                  /* Set lengths of packet application data */
                  dataLength[0] = 2000;
                  dataLength[1] = 3000;
                  dataLength[2] = 4000;

                  /* Fill appData buffer as desired here.

                  Do not include packet header data—it is filled by this
                        tool.

Fill first 2000 bytes with first packet data,

next 3000 bytes with second packet data,

last 4000 bytes with third packet data */

/* Create simulated file */

                  returnStatus =
                    PGS_IO_L0_File_Sim(
                        spacecraftTag,
                        appID,
                        firstPacketNum,
                        startUTC,
                        numValues,
                        timeInterval,
                        dataLength,
                        otherFlags,
                        filename,
                        appData,
                        qualMissLen,
                        qualData,
                        missData,
                        );
```

FORTRAN:

```
      implicit none

      integer pgs_io_l0_file_sim

      integer spacecraftTag
      integer appid(3)
      integer firstpacketnum
      character*27 startutc
      integer numvalues
      double precision timeinterval
      integer datalength(3)
      integer otherflags(2)
      character*256 filename
      character*9000 appdata
      integer qualmisslen(2)
      character*28 qualdata
      character*16 missdata

      integer returnstatus

      spacecraftTag = TRMM
      appid(1) = 54
      appid(2) = 54
      appid(3) = 54
      firstpacketnum = 1
      startutc = ‘1994-12-31T12:00:00.000000’
      numvalues = 3
      timeinterval = 6.6

C   Set lengths of packet application data
      datalength(1) = 2000
      datalength(2) = 3000
      datalength(3) = 4000

C   Fill data to write to file header
      otherflags(1) = 68 ! Redundant Data Deleted & RS Decoding
      otherflags(2) =  1 ! Routine production data
```

```
                  filename = 'TRMM_G0088_1997-12-01T00:00:00Z_V01.dataset_01'
                  qualmisslen(1) = 28
                  qualmisslen(2) = 16

C   Fill appData buffer as desired here.

C   Do not include packet header data—it is filled by this tool.

C   Fill first 2000 bytes with first packet data,

C       next 3000 bytes with second packet data,

C       last 4000 bytes with third packet data

C   Create simulated file

returnstatus = pgs_io_l0_file_sim(

                                                 spacecrafttag,
                                                 appid,
                                                 firstpacketnum,
                                                 startutc,
                                                 numvalues,
                                                 timeinterval,
                                                 datalength,
                                                 filename,
                                                 otherflags
                                                 appdata,
                                                 qualmisslen,
                                                 qualdata,
                                                 missdata)
```

NOTES: This tool is intended for use in science software development and testing, but not for production purposes.

When used to create file for EOS AM (EDOS format) the Construction Record creation tool (PGS_IO_L0_EDOS_hdr_Sim()) must also be called to create the PDS/EDS Construction Record.

RELEASE NOTES:

This function conforms to EDOS-EGS ICD (June 28, 1996)

See Section 6.2.1.1.6 Open Issues

REQUIREMENTS: There is no SDP Toolkit requirement for this functionality. This tool was created to support internal ECS SDP Toolkit development and testing, and it is being provided as a service to the user.

6.2.1.2 HDF File I/O Tools

The ECS standard file format for transmission of datasets is National Center for Supercomputer Application's (NCSA's) Hierarchical Data Format (HDF). ECS has built extensions to NCSA HDF, known as HDF-EOS, which will support most recognized EOS era earth sciences data structures. Presently these data structures are grid, point and swath structures. If, in some cases, these are not sufficient, NCSA HDF could be used along with ECS metadata to specify an output file. Version 2.0 of HDF-EOSis delivered with SCF Toolkit 5.2.

HDF-EOS is built on HDF low level functions and NCSA conventions were adhered to. The most prominent example is the user input of physical file handles. HDF requires physical handles, while the SDP toolkit requires logical handles. In order to make the toolkit compatible with HDF, the user will make one additional call to a process control function, obtain a physical handle and then open an HDF (HDF-EOS) file. Toolkit error handling functions may be used as necessary or desired. See the example in this section.

Important: HDF was designed to be a transport file format only, and support for such endeavors as updating a pre-existing file is very weak. Because of this and other performance considerations, HDF may not be the best choice of file format to use in internal processing of your files. We therefore strongly recommend that you use the Generic (Section 6.2.1.3) and Temporary (Section 6.2.1.6) I/O functions for internal processing, and reserve the use of HDF for initial read and final write of data products meant for archival and distribution.

EXAMPLE OF USAGE OF NCSA HDF FUNCTIONS

The following code fragments are simple examples of how the science software might use the SDP Toolkit logical-to-physical filename translation function in conjunction with the NCSA HDF open function. See Sections 6.2.2, 6.2.3, Appendices C and B.

The examples assume the following exists in the Process Control File (PCF):

? PRODUCT OUTPUT FILES

399|test10.hdf|/fire2/toma/data||||3

399|test9.hdf|/fire2/toma/data||||2

399|test8.hdf|/fire2/toma/data||||1

C

```
#include <PGS_PC.h>
#include <hdf.h>
#include <dfi.h>
#define HDF_INFILE 399
PGSt_integer version;
char physical_filename[PGSd_PC_FILE_PATH_MAX];
PGSt_SMF_status returnStatus;
int32 hdf_status;
int16 n_dds;
/*
```

```
Begin example
*/
version = 1;
returnStatus = PGS_PC_GetReference
      ( HDF_FILE, &version, physical_filename );
/*
Variable physical_filename now contains the string
“/fire2/toma/data/test10.hdf”
Variable version now contains the value 2, i.e., the number
of versions left in order, below this version in the PC file
*/
/*
Open the HDF file
*/
n_dds = 5; /* No. HDF data descriptor blocks */
hdf_status = Hopen(physical_filename,DFACC_CREATE,n_dds);
```

FORTRAN:

```
  implicit none

  INCLUDE           ‘PGS_SMF.f’
  INCLUDE           ‘PGS_PC.f’
  INCLUDE           ‘PGS_PC_9.f’
  INTEGER           HDF_INFILE
  PARAMETER         (HDF_INFILE=399)
  CHARACTER*(*)     physicalfilename
  INTEGER           pgs_pc_getreference
  INTEGER           version
  INTEGER           returnstatus
  INTEGER           hdfstatus
  INTEGER           ndds

 C

 C Begin example

 C

  version = 1
        returnstatus = pgs_pc_getreference
        .           ( HDF_INFILE, version, physicalfilename )

C

C Variable physicalfilename now contains the string

C “/fire2/toma/data/test10.hdf”

C Variable version now contains the value 2, i.e., the number
```

```
C of versions left in order below this version in the PC file
C
C Open the HDF file
C
   ndds = 5    ! No. HDF data descriptor blocks
   hdfstatus = hopen(physicalfilename,DFACC_CREATE,ndds)
```

NOTES:

a. In order for this tool to work properly in the SCF environment, a Process Control File (PCF) must first be created by the science software developer. This file is part of the mechanism that maps the logical file identifiers in the science code to physical filenames. (This mapping will be performed by the scheduling subsystem in the DAAC environment.) See Section 4.2.2, "File Management," for further discussion. UNIX environment variable $PGS_PC_INFO_FILE must point to the this file.

   In general, the PCF created by the user must follow the format given in Appendix C.

b. Currently, the Toolkit installation script installs HDF 4.1r1.

c. Functions that write error messages to a log file are now available. See the Status Message (SMF) tool section.

6.2.1.3 Generic File I/O Tools

This section includes tools for performing I/O functions on files that are not in the ECS standard format, i.e., HDF. The file open tools (Gen_Open and Gen_OpenF) are used by the science software to open miscellaneous files, which means any files that are not HDF, Level 0, ancillary, temporary or intermediate files (see sections 6.2.1.2, 6.2.1.1, 6.3.1, and 6.2.1.6). The file close tools (Gen_Close and Gen_CloseF) are used in science software to close these miscellaneous files, and also to close temporary and intermediate files.

The tools in this section are also used by other Toolkit functions, to access ancillary files (section 6.3.1), Level 0 files (section 6.2.1.1) and other miscellaneous files.

There are three items that apply to this entire subgroup of tools:

a. These tools only perform open and close functions on files. Reads, writes and other I/O functions are to be done by native C and FORTRAN I/O.

b. Due to file handle and other considerations it was not possible to bind FORTRAN to the C tools using the macro binding package. Unlike the rest of the Toolkit, these functions have separate FORTRAN versions.

c. Science software should use the PGS_IO_Temp_Open tool to open a temporary or intermediate file; see Section 6.2.1.6.

Special note regarding FORTRAN 90: Tools PGS_IO_Gen_OpenF and PGS_IO_Gen_Temp_OpenF now have FORTRAN 90 versions. These versions support two specific usages of the F90 OPEN function that are not supported in ANSI FORTRAN 77; they do not support all F90 options of OPEN. At Toolkit installation time, you select between F77 and F90, and the appropriate source code file is compiled; the function names are the same in both versions of FORTRAN. Options and text that apply only to FORTRAN 90 are marked in this document as ***F90 SPECIFIC***.

Open a Generic File (C Version)

NAME: **PGS_IO_Gen_Open( )**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_Gen_Open(
        PGSt_PC_Logical            file_logical,
        PGSt_IO_Gen_AccessType     file_access,
        PGSt_IO_Gen_FileHandle     **file_handle,
        PGSt_integer               file_version)
```

FORTRAN: (not applicable)

DESCRIPTION: Upon a successful call, this function will provide the argument PGSt_IO_Gen_FileHandle to support other "C" library stream manipulation routines.

INPUTS: file_logical-User defined logical file identifier

file_access-type of access granted to opened file:

Table 6-8. File Access Type

Toolkit	C	Description
PGSd_IO_Gen_Read	"r"	Open file for reading
PGSd_IO_Gen_Write	"w"	Open file for writing, truncating existing file to 0 length, or creating a new file
PGSd_IO_Gen_Append	"a"	Open file for writing, appending to the end of existing file, or creating file
PGSd_IO_Gen_Update	"r+"	Open file for reading and writing
PGSd_IO_Gen_Trunc	"w+"	Open file for reading and writing, truncating existing file to zero length, or creating new file
PGSd_IO_Gen_Append Update	"a+"	Open file for reading and writing, to the end of existing file, or creating a new file; whole file can be read, but writing only appended

file_version-specific version of the logical file. (NOTE: this value will default to '1' for the interim delivery. Multiple file versions will be a capability in Toolkit 3)

OUTPUTS: file_handle-used to manipulate files with other "C" library stream I/O routines

RETURNS:

Table 6-9. PGS_IO_Gen_Open Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX system error
PGSIO_E_GEN_OPENMODE	Invalid access mode
PGSIO_E_GEN_FILE_NOEXIST	No entry for file logical in $PGS_PC_INFO_FILE
PGSIO_E_GEN_REFERENCE_FAILURE	Other error accessing $PGS_PC_INFO_FILE
PGSIO_E_GEN_BAD_ENVIRONMENT	Environment error reported by Process Control

(NOTE: the above are short descriptions only; full text of messages appears in files $PGSMSG/PGS_IO_1.t. Descriptions may change in future releases depending on external ECS design.)

EXAMPLE:

```
// This example illustrates how to open a Product Output
   File for writing //

PGSt_SMF_status         returnStatus;
PGSt_PC_Logical         logical;
PGSt_IO_Gen_AccessType  access;
PGSt_IO_Gen_FileHandle  *handle;
PGSt_integer            version;

logical = 10;
version = 1;        // will default to 1 for Toolkit 3 on out //
access = PGSd_IO_Gen_Write;
returnStatus = PGS_IO_Gen_Open( logical,access,&handle,
                                version );
if (returnStatus != PGS_S_SUCCESS)
{
    goto EXCEPTION;
}
    .
    .
    .
EXCEPTION:
```

NOTES:

A file opened for write that already exists will be overwritten.

This function will support all POSIX modes of fopen.

While all modes of access are supported for this tool, those modes that allow for writing to a file (i.e., not PGSd_IO_Gen_Read) are intended for Toolkit access only. The only files that the science software should write to are product output files (HDF) and Temporary, or Intermediate files. The only exceptions to this are for Support Output files that may need to be archived, but which are not considered to be products.

!!!!!!!!!!! ALERT !!!!!!!!!!!

During testing of this tool, the mode AppendUpdate (a+)!! was found to produce results that were not consistent with the documented POSIX standard. The sort of behavior that was typically observed was for data, buffered during a read operation, to be appended to the file along with other data that was being written to the file. Note that this behavior could not be attributed to the Toolkit since the same behavior was revealed when purely "POSIX" calls were used.

IMPORTANT TOOLKIT 5 NOTES

The following environment variable MUST be set to assure proper operation:

PGS_PC_INFO_FILE path to process control file

However, the following environment variables are NO LONGER recognized by the Toolkit as such:

PGS_PRODUCT_INPUT	path to standard input files
PGS_PRODUCT_OUTPUT	path to standard output files
PGS_SUPPORT_INPUT	path to supporting input files
PGS_SUPPORT_OUTPUT	path to supporting output files

Instead, the default paths, which were defined by these environment variables in previous Toolkit releases, may now be specified as part of the Process Control File (PCF). Essentially, each has been replaced by a global path statement for each of the respective subject fields within the PCF. To define a global path statement, simply create a record that begins with the '!' symbol defined in the first column, followed by the global path to be applied to each of the records within that subject field. Only one such statement can be defined per subject field and it must be appear prior to any dependent subject entry.

The status condition PGSIO_E_GEN_BAD_ENVIRONMENT now indicates an error status on the global path statement as defined in the PCF, and NOT on an environment variable. However, as with previous releases, the status message associated with this condition may reference the above "tokens," but this is only to indicate which of the global path statements is problematic.

REQUIREMENTS: PGSTK-0360, PGSTK-1360

Open a Generic File (FORTRAN Version)

NAME: **PGS_IO_Gen_OpenF( )**

SYNOPSIS:

C: (not applicable)

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'

integer function pgs_io_gen_openf(file_logical, file_access,
                                  record_length, file_handle,
                                  file_version)

integer file_logical
integer file_access
integer record_length
integer file_handle
integer file_version
```

DESCRIPTION: Upon a successful call, this function will allocate a logical unit number to support FORTRAN READ and WRITE statements. This is returned to the user via the parameter file_handle. The user provides the logical file identifier and file version number, which internally get mapped to the associated physical file.

INPUTS: file_logical-User defined logical file identifier

file_access-type of access granted to opened file:

Table 6-10. File Access Type (1 of 2)

PGS FORTRAN Access Mode	Rd/Wr/Update/ Append	FORTRAN 77/90 'access='	FORTRAN 77/90 'form='
PGSd_IO_Gen_RSeqFrm	Read	Sequential	Formatted
PGSd_IO_Gen_RSeqUnf	Read	Sequential	Unformatted
PGSd_IO_Gen_RDirFrm	Read	Direct	Formatted
PGSd_IO_Gen_RDirUnf	Read	Direct	Unformatted
PGSd_IO_Gen_WSeqFrm	Write	Sequential	Formatted
PGSd_IO_Gen_WSeqUnf	Write	Sequential	Unformatted
PGSd_IO_Gen_WDirFrm	Write	Direct	Formatted

Table 6-10. File Access Type (2 of 2)

PGS FORTRAN Access Mode	Rd/Wr/Update/ Append	FORTRAN 77/90 'access='	FORTRAN 77/90 'form='
PGSd_IO_Gen_WDirUnf	Write	Direct	Unformatted
PGSd_IO_Gen_USeqFrm	Update	Sequential	Formatted
PGSd_IO_Gen_USeqUnf	Update	Sequential	Unformatted
PGSd_IO_Gen_UDirFrm	Update	Direct	Formatted
PGSd_IO_Gen_UDirUnf	Update	Direct	Unformatted
F90 SPECIFIC			
PGSd_IO_Gen_ASeqFrm	Append	Sequential	Formatted
PGSd_IO_Gen_ASeqUnf	Append	Sequential	Unformatted

record_length-record length must be greater than 0 for direct access

F90 SPECIFIC must be greater than or equal to 0 for sequential access; if 0, file is opened with default record length

file_version-version of file to open (minimum value = 1)

OUTPUTS: file_handle-used to manipulate files READ and WRITE

RETURNS:

Table 6-11. PGS_IO_Gen_OpenF Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_NO_FREE_LUN	All logical unit numbers are in use
PGSIO_E_GEN_OPENMODE	Illegal open mode was specified
PGSIO_E_GEN_OPEN_OLD	Attempt to open with STATUS=OLD failed
PGSIO_E_GEN_OPEN_NEW	Attempt to open with STATUS=NEW failed
PGSIO_E_GEN_OPEN_RECL	Invalid record length specified
PGSIO_E_GEN_FILE_NOEXIST	File not found, cannot create
PGSIO_E_GEN_REFERENCE_FAILURE	Can't do Temporary file reference

EXAMPLE:

```
integer     returnstatus
integer     file_logical
integer     file_access
integer     record_length
integer     file_handle
integer     file_version

file_version= 3
file_logical= 101
file_access       = PGSd_IO_Gen_WSeqFrm
```

```
                  returnstatus = PGS_IO_Gen_OpenF( file_logical, file_access,
                                                   record_length, file_handle,
                                                   file_version)
if (returnstatus .NE. PGS_S_SUCCESS) then
                 C    goto 1000
                  end if
                    .
                    .
                    .
1000 <error handling goes here>
```

NOTES: While all modes of access are supported for this tool, those modes that allow for writing to a file (i.e., not PGSd_IO_Gen_Read) are intended for Toolkit access only. The only files that the science software should write to are product output files (HDF) and Temporary, or Intermediate files.

In order to ascertain the number of versions currently associated with the logical identifier in question, make a call to PGS_PC_Get_NumberOfFiles() first (Toolkit 3 and later.)

Due to the nature of FORTRAN IO, it is possible to write a file opened for reading as well as read a file opened for writing. The matching of access mode to IO statement cannot be enforced by the tool. This is up to the user.

Once a file has been opened with this tool, it must be closed with a call to PGS_IO_Gen_CloseF before being re-opened. Failure to do this will result in undefined behavior.

IMPORTANT TOOLKIT 5 NOTES

The following environment variable MUST be set to assure proper operation:

PGS_PC_INFO_FILEpath to process control file

However, the following environment variables are NO LONGER recognized by the Toolkit as such:

PGS_PRODUCT_INPUT	path to standard input files
PGS_PRODUCT_OUTPUT	path to standard output files
PGS_SUPPORT_INPUT	path to supporting input file
PGS_SUPPORT_OUTPUT	path to supporting output files

Instead, the default paths, which were defined by these environment variables in previous Toolkit releases, may now be specified as part of the Process Control File (PCF). Essentially, each has been replaced by a global path statement for each of the respective subject fields within the PCF. To define a global path statement, simply create a record that begins with the ‘!’ symbol defined in the first column, followed by the global path to be applied to each of the records within that subject field. Only one such statement can be defined per subject field and it must be appear prior to any dependent subject entry.

It is error condition to have an input file specified in the PCF that does not exist on disk. The behavior of the tool is undefined when attempting to open such a file for reading.

REQUIREMENTS: PGSTK-0360

Close a Generic File, Temporary or Intermediate File (C Version)

NAME: **PGS_IO_Gen_Close( )**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_Gen_Close(
        PGSt_IO_Gen_FileHandle    *file_handle);
```

FORTRAN: (not applicable)

DESCRIPTION: This tool closes a stream opened by a call to the “C” version of the Generic I/O Open tools.

INPUTS: fileHandle-file handle returned by PGS_IO_Gen_Open or PGS_IO_Gen_Temp_Open.

OUTPUTS: NONE

RETURNS:

Table 6-12. PGS_IO_Gen_Close Returns

Return	Description
PGS_S_SUCCESS	Success
PGSIO_E_GEN_CLOSE	Error closing file

EXAMPLES:

```
PGSt_IO_Gen_FileHandle  *handle;
PGSt_SMF_status         returnStatus;

returnStatus = PGS_IO_Gen_Close( handle );
if (returnStatus != PGS_S_SUCCESS)
{
   goto EXCEPTION;
}
else
{
   .
   .
   .
}
EXCEPTION:
```

NOTES:	Usage of this tool is optional, but failure to close a file could result in loss of data, destroyed files, or possible intermittent errors in your program.
	As a consequence of calling this tool, any data left unwritten in the output buffer will be flushed to the output stream; likewise, any data left unread in the input buffer will be discarded.
	!!!!!!!!!! ALERT !!!!!!!!!!!
	Never attempt to close a file that has not been initialized, or previously used in an open call. Failure to heed this warning will result in program abort on many platforms.
REQUIREMENTS:	PGSTK-0360

Close a Generic File (FORTRAN Version)

NAME: **PGS_IO_Gen_CloseF( )**

SYNOPSIS:

C: (not applicable)

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'

integer pgs_io_gen_closef(file_handle)
integer file_handle
```

DESCRIPTION: This tool closes a FORTRAN IO unit opened by call to PGS_IO_Gen_OpenF or PGS_IO_Gen_Temp_OpenF.

INPUTS: file_handle-file handle returned by PGS_IO_Gen_OpenF or PGS_IO_Gen_Temp_OpenF

OUTPUTS: NONE

RETURNS:

Table 6-13. PGS_IO_Gen_CloseF

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_GEN_CLOSE	Attempt to close file failed
PGSIO_E_GEN_ILLEGAL_LUN	file_handle LUN was out-of-bounds
PGSIO_W_GEN_UNUSED_LUN	file_handle LUN was not in use

EXAMPLES:

```
integer     handle
integer     returnstatus

returnstatus = PGS_IO_Gen_CloseF(handle)
if (returnstatus != PGS_S_SUCCESS) goto 1000
.
.
.
100 <error handling goes here>
```

NOTES: Failure to close a file could result in loss of data, destroyed files, or possible intermittent errors in your program.

This tool expects the input file_handle to point to a file that was successfully opened via a call to either the tool PGS_IO_Gen_OpenF or the tool PGS_IO_Gen_Temp_OpenF. If this is not the case, the result of calling the tool is undefined.

REQUIREMENTS: PGSTK-0360

6.2.1.4 Metadata Tools

This set of tools is designed to manage the metadata that are generated with each EOS product, i.e., the granule-level metadata. The tools also provide a mechanism for populating the inventory data base tables with the metadata for each granule. The purpose of these tools is:

- To ensure that the metadata produced conforms to ECS standards in content and format; and
- To provide access files from within the science algorithms to metadata contained in input files.

The overall context of metadata in ECS, and further details on the use of the metadata tools are provided in Appendix J of this document.

The metadata tools in the SDP toolkit library are called from within a PGE to read and write metadata. The metadata attributes that will be assigned values during processing are identified in the metadata configuration file (MCF). The MCF is read into memory and toolkit calls are used to populate values for the attributes. When the metadata population process is complete, metadata “blocks” are written to product output files as HDF data objects called global attributes (not to be confused with individual metadata elements which are also called attributes). All output metadata is in object description language (ODL).

Multiple MCFs may be opened and written to from within a single PGE. The five metadata tools that are used in conjunction with MCFs must be called in a specific sequence, once for each MCF. First, each MCF must be initialized with **PGS_MET_Init**, which also assigns values for “system” metadata. Values generated within the PGE are assigned to attributes in the MCF using **PGS_MET_SetAttr**. To return the value of any metadata attribute in the MCF that has received a value **PGS_MET_GetSetAttr** may be used. After all values have been assigned, **PGS_MET_Write** is used to write the metadata to the product or, alternatively for non-HDF products, to a separate ASCII metadata file. Finally, **PGS_MET_Remove** frees up memory used by the MCFs .

Two additional toolkit routines are used to read metadata values from within the PGE. These may be called independently of any MCF. **PGS_MET_GetPCAttr** may be used to return the value of metadata from input files identified to the process control (PC) system. **PGS_MET_GetConfigData** may be used to return the value of runtime metadata from the Process Control File.

The FORTRAN versions of PGS_MET_SetAttr, PGS_MET_GetConfigData, PGS_MET_GetSetAttr, and PGS_MET_GetPCAttr must include an underscore and an extra character at the end of the function name to indicate the data type being handled: _S for string values, _I for integer and unsigned int values, and _D for single or double precision real values. For example, the function PGS_MET_SetAttr actually represents three different FORTRAN functions:

- PGS_MET_SetAttr_S to set the value of string and datetime attributes

- PGS_MET_SetAttr_I to set integer and unsigned int values; and
- PGS_MET_SetAttr_D to set real or double values

As discussed in greater detail in Appendix J, two separate metadata blocks are handled by the metadata tools. These are called inventory and archive. Inventory consists of “core” attributes, i.e. those that are part of the ECS Data Model, which will reside in the ECS inventory tables and will thus be available to query on in locating granules. Archive metadata refers to metadata that a data producer wants to be included with the data granule, but need not be searchable by the system and will therefore not be used to populate the inventory tables. Archive metadata can, however, be read from HDF input files using toolkit calls.

The inventory and archive blocks are referenced in the toolkit calls by an array, e.g. mdHandles(n), where n=1 (for C, n=2 for FORTRAN) indicates inventory metadata and n=2 (or n=3 for FORTRAN) indicates archive metadata. To write an ASCII version of the metadata for non-HDF files mdHandles(0) (or n=1 for FORTRAN) is used to indicate that all metadata block are to be written together. It is possible to define other blocks and write them to HDF product output files or to ASCII metadata output files, but these will not be handled by the system. For example, if the granule is subsetted using ECS routines, only the inventory and archive blocks will be copied into the resultant file.

Additional description and extensive examples of the usage of MET tools can be found in the *HDF-EOS Users Guide for the ECS Project, Vol. 1*, Section 7 and 8.

A description of each MET tool follows:

Initialize a Metadata Configuration File (MCF) into Memory

NAME: **PGS_MET_Init()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_Init(
        PGSt_PC_Logical          fileId,
        PGSt_MET_all_handles     mdHandles)
```

FORTRAN:

```
include "PGS_MET_13.f"
include "PGS_MET.f"
include  "PGS_SMF.h"

integer function pgs_met_init(fileId, mdHandles)

integer         fileId
character*      PGS_MET_GROUP_NAME_L
                mdHandles(PGS_MET_NUM_OF_GROUPS)
```

DESCRIPTION: Initializes MCF file containing metadata.

INPUTS:

Table 6-14. PGS_MET_Init Inputs

Name	Description	Units	Min	Max
fileId	MCF file id	none	variable	variable

OUTPUTS:

Table. 6-15. PGS_MET_Init Outputs

Name	Description	Units	Min	Max
mdHandles	metadata groups in MCF	none	N/A	N/A

RETURNS:

Table. 6-16. PGS_MET_Init Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_LOAD_ERR	Unable to load <MCF> information. Lower level routines contain more information
PGSMET_E_GRP_ERR	Master groups are not supposed to be enclosed under any other group or object. The offending group is <name>
PGSMET_E_GRP_NAME_ERR	Group name length should not exceed PGS_MET_GROUP_NAME_L - 5.
PGSMET_E_NO_INVENT_DATA	Inventory data section not defined in the MCF
PGSMET_E_DUPLICATE_ERR	There is a another object with the same name for object <name> Duplicate names are not allowed within master groups
PGSMET_E_NUM)FMCF_ERR	Unable to load. The number of MCFs allocated has been exceeded.
PGSMET_E_PCF_VALUE_ERR	Metadata objects to be set from values defined in PCF could not be set. See error returns form the lower level routines. Initialization takes place nevertheless.

EXAMPLES:

C:

```
#include "PGS_MET.h"
#define     INVENTORYMETADATA 1
#define  MODIS_FILE 10253 /* This value must also be  defined in the PCF
      10253|hdftestfile|/home/asiyyid/pgetest/fortran/|||hdf
      testfile|1  :   */

#define  ODL_IN_MEMORY 0
int main()
{
PGSt_MET_all_handles handles;
char * fileName = "/home/modis/hdftestfile"; /* the               user
should change this accordingly */
int32   hdfRet, sdid;
extern AGGREGATE PGSg_MET_MasterNode;
PGSt_SMF_status  ret = PGS_S_SUCCESS;
PGSt_integer    fileId = PGSd_MET_MCF_FILE;
PGSt_integer i;
double   dval, dval[6];
char* sval;
sval = (char*) malloc(30);
ret= PGS_MET_Init(fileId, handles);
if(ret != PGS_S_SUCCESS)
```

```
                    {
            printf("initialization failed\n");
            return 0;
                    }

            PGS_MET_Remove();
            printf("SUCCESS\n");
            return 0;
            }
```

FORTRAN:

```
      include "PGS_SMF.f"
      include "PGS_MET_13.f"
      include "PGS_MET.f"
C     the file id must also be defined in the PCF as follows
C     10253|hdftestfile|/home/asiyyid/pgetest/fortran/|||hd
C     testfile|1
            integer pgs_met_init
      integer MODIS_FILE
      parameter(MODIS_FILE = 10253)
            integer INVENTORYMETADATA
      parameter(INVENTORYMETADATA = 2)
      integer ODL_IN_MEMMORY
      parameter(ODL_IN_MEMMORY = 1)
C     the groups have to be defined as 49 characters long.
C     The C interface is 50.
C     The cfortran.h mallocs an extra 1 byte for the null
C     character '\0/', therefore making the actual length of a
C     string pass as 50.
      character*PGS_MET_GROUP_NAME_L
    1 mdHandles(PGS_MET_NUM_OF_GROUPS)
      character*50 fileName
      integer   result
      integer   pgs_met_init
      integer   hdfReturn
      double precision dval(1), dval(6)
      char*80   sval(5)
C     you must change this file spec in the PCF and the example
C     before running this example.
      fileName = "/home/asiyyid/pgetest/fortran/hdftestfile"
      result = pgs_met_init(PGSd_MET_MCF_FILE, groups)
      if(result.NE.PGS_S_SUCCESS) then
      print *, "Initialization error. See Logstatus for details"
      endif
      print *, "SUCCESS"
      end
```

NOTES: The MCF file must be in the format described in Appendix J.

Effective with the November 1996 SCF Toolkit release, multiple MCFs can now be initialized by repeated calls to this function.

REQUIREMENTS: PGSTK-0290, PGSTK-0370

Assign Values to Metadata Attributes

NAME: **PGS_MET_SetAttr()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status

PGS_MET_SetAttr(
        PGSt_MET_handle    mdHandle,
        char               *attrNameStr,
        void               *attrValue)
```

FORTRAN:

```
include "PGS_MET_13.f"
        include "PGS_MET.f"
        include "PGS_SMF.h"

integer function pgs_met_setattr(mdHandle, attrNameStr, attrValue)

character*(*)       mdHandle
character*(*)       attrName
'user defined'      attrValue
```

DESCRIPTION: After an MCF file is initialized into memory the user may assign values to metadata attributes using PGS_MET_SetAttr(). The values can be of following types and their array counterparts

PGSt_integer, PGSt_double, PGSt_real, char * (string)

INPUTS:

Table 6-17. PGS_MET_SetAttr Inputs

Name	Description	Units	Min	Max
mdHandle	metadata group in MCF	none	N/A	N/A
attrNameStr	name.class of parameter	none	N/A	N/A
attrValue	value of attribute to be inserted	none	N/A	N/A

OUTPUTS: None

RETURNS:

Table 6-18. PGS_MET_SetAttr Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_NO_INITIALIZATION	Metadata file is not initialized
PGSMET_E_NESTED_OBJECTS	Object descriptions enclosing related objects must not be enclosed themselves by other objects
PGSMET_E_ODL_MEM_ALLOC	ODL routine failed to allocate memory
PGSMET_E_PARENT_GROUP	Multiple objects must have enclosing groups around them
PGSMET_E_CLASS_PARAMETER	Container object must also have class parameter defined
PGSMET_E_METADATA_CHILD	metadata Objects are not allowed to enclose other objects
PGSMET_W_NOT_MULTIPLE	Object is not supposed to be multiple therefore resetting the value. The user may have given a class with the metadata name
PGSMET_E_ILLEGAL_HANDLE	Handle is illegal. Check that initialization has taken place.
PGSMET_E_ILLEGAL_TYPE	Illegal type definition for metadata <attrName>. It should be a string
PGSMET_E_NO_DEFINITION	Unable to obtain <attr> of metadata <parameter> Either type or numval not defined
PGSMET_E_ILLEGAL_NUMVAL	Illegal NUMVAL definition for metadata <attrName>. It should be an integer
PGSMET_E_DD_UNKNOWN_PARM	The requested parameter <parameter name> could not be found in <agg node>
PGSMET_E_NEW_ODL_DATA_ERR	Unable to create a new odl <parameter>, probably due to lack of memory
PGSMET_E_INV_DATATYPE	Invalid data type definition in MCF for parameter <name>
PGSMET_E_INVALID_LOCATION	Invalid location for setting attribute value

EXAMPLES:

C:

```
/* For setting Inventory Attributes in the MCF */

/* NUMVAL i the MCF = 6 */

            dvals[0] = 10.0;
            dvals[1] = 20.0;
            dvals[2] = 30.0;
            dvals[3] = 40.0;
            dvals[4] = 50.0;
            dvals[5] = 60.0;
            ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
                  "GRingPointLatitude.1", dvals);
```

```
 /* For setting Product Specific Attributes */

strcpy(informationname,"TestingAttribute1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
"AdditionalAttributeName.1",&informationname);
strcpy(informationname,"testingAttributeValue1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
```

FORTRAN:

```
 C  For setting Inventory Attributes in an HDF file

            dvals(1) = 10.0
            dvals(2) = 20.0
            dvals(3) = 30.0
            dvals(4) = 40.0
            dvals(5) = 50.0
            dvals(6) = 60.0
            ret =
            pgs_met_setattr_d(groups(INVENTORYMETADATA),
     1"GRingPointLatitude.1", dvals)

C  For setting Product Specific Attributes

            informationname = "TestingAttribute1"
            ret = pgs_met_setattr_s(groups(INVENTORYMETADATA),
     1"AdditionalAttributeName.1",informationname)
            informationname = "testingAttributeValue1"
            ret = pgs_met_setattr_s(groups(INVENTORYMETADATA),
     1"ParameterValue.1",informationname)
```

NOTES: **1. Multiplicity:**

In TK5, a CLASS statement was introduced so that metadata objects with the same name could be distinguished from each other in the ODL tree. In TK5.1 this functionality was further extended to allow a single metadata object in the MCF to have multiple instances. This means that all the metadata objects within a master group in the MCF must have unique names. The use of the CLASS field in the name of a metadata attribute is optional and is needed only when the attribute in the MCF is within a group having a CLASS statement. See Appendix J for details and examples.

2. Nested Metadata:

There are certain metadata objects which are always described as a group of related metadata. To allow such groups to stay together in the MCF and the ODL tree, nested metadata objects are defined in the MCF using "Container Objects." in the MCF with related metadata as its child members. The child members are set individually as before. The container object does not have a value since it defines a concept and not an entity.

In the case of multiple container objects (e.g. there could be more than one instances of gring polygons), when a call to set a value of one of the child metadata objects is made, it is the container object which is duplicated with a different class creating instances of all the child members. It is the users responsibility to set their values as well with subsequent call. Examples are given in Appendix J.

3. Array Filling:

TK5 imposed a restriction that metadata objects with values defined as arrays must be set with all the elements filled. This restriction is now lifted and the user has the freedom to set 1 to n values for a particular parameter where n is defined in the NUM_VAL field in the MCF. In this case where the values are being retrieved, the end of array is marked by:

INT_MAX	for integers
UINT_MAX	for unsigned integers
DBL_MAX	for doubles
NULL	char * (strings)

These values are defined in the limits.h and floats.h. Its analogous to null terminated strings defined as char[] arrays.

FORTRAN Users:

Use PGSd_MET_INT_MAX, PGSd_MET_DBL_MAX and PGSd_MET_STR_END respectively.

The user can check for these values to determine the actual number of values retrieved. In case where the number of values retrieved is equal to n, there is no end of array marker since user is expected to know n for setting the return buffer.

4. Permissible Data Locations:

PGS_MET_SetAttr can be used to assign values to metadata attributes which have DATA_LOCATION = “PGE”, “MCF”, “PCF”, or “TK”. Any attribute with DATA_LOCATION = “DSS”, “DAAC,” or “DP” can not be set by the PGE. An attempt to do so with PGS_MET_SetAttr will result in an error message of PGSMET_E_INVALID_LOCATION being generated in the runtime LOG file.

5. Metadata Types:

The tool provides a void interface through which different types of metadata can be set. The types supported are:

PGSt_integer
PGSt_uinteger
PGSt_double
string

and their arrays counterparts. PGSt_real has been omitted because of the changes in TK5.1.

It is very important that variable string pointers are used for string manipulations. This is because void interface is used. For example, the following piece of code would give an error or unexpected results:

```
.
.
char a[100];
.
.
strcpy(a, "MODIS");
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],
"SATELLITE_NAME", a);
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],
"SATELLITE_NAME", &a);
```

The first call is wrong because the routine expects char** but cannot force it because of void interface. The second call is wrong too because of the declaration of 'a' which is a constant pointer, i.e. it would always point to the same location in memory of 100 bytes. Only the following construct will work with the routine in which the string pointer is declared as a variable

```
char *a = "MODIS"
.
.
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],
"SATELLITE_NAME", &a);
```

The above discussion is also true for arrays of strings. For example, the following is not allowed for the same reasons as above

```
.
.
char a[10][100];
.
```

```
.
strcpy(a[0], "MODIS");
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],
"SATELLITE_NAME", &a[0]);
```

while the following is acceptable:

```
.
.
char *a[10];
.
.
a[0] = "MODIS";
```

retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA], "SATELLITE_NAME", &a[0]);

IMPORTANT

The void buffer should always be large enough for the returned values otherwise routine behavior is uncertain.

REQUIREMENTS: PGSTK-0290 PGSTK-0410 PGSTK-380

Accesses Metadata Attributes Already Set in Memory

NAME: **PGS_MET_GetSetAttr()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_GetSetAttr(
PGSt_MET_handle    mdHandle,
          char*          attrNameStr,
          void*          attrValue)
```

FORTRAN:

```
include        "PGS_MET_13.f"
include        "PGS_MET.f"
include        "PGS_SMF.h"

integer function pgs_met_getsetattr(mdHandle, attrNameStr, attrValue)
character*     mdHandle
character*     attrName
'user defined' attrValue
```

DESCRIPTION: The MCF is first initialized into memory and some of the parameters are automatically set and some are set by the user using PGS_MET_SetAttr(). This tool is used to retrieve these values.

INPUTS:

Table 6-19. PGS_MET_GetSetAttr Inputs

Name	Description	Units	Min	Max
mdHandle	metadata group	none	N/A	N/A
attrName	name.class of parameter	none	N/A	N/A

OUTPUTS:

Table 6-20. PGS_MET_GetSetAttr Outputs

Name	Description	Units	Min	Max
attrValue	value of attribute to be passed back to the user	none	N/A	N/A

RETURNS:

Table 6-21. PGS_MET_GetSetAttr Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_NO_INITIALIZATION	Metadata file is not initialized
PGSMET_E_DD_UNKNOWN_PARM	The requested parameter <parameter name> could not be found in <agg node>
PGSMET_W_METADATA_NOT_SET	The metadata <name> is not yet set
PGSMET_E_NO_DEFINITION	Unable to obtain <attr> of metadata <parameter>
	Either NUM_VAL or type is not defined
PGSMET_E_ILLEGAL_HANDLE	Handle is illegal. Check that initialization has taken place.

EXAMPLES:

C:

```
/* For accessing Inventory Attributes in an HDF file */

            for(i = 0; i < 6; i++) dvals[i] = 0.0;
            ret = PGS_MET_GetSetAttr(handles[INVENTORYMETADATA],
            "GRingPointLatitude.1", dvals);
            for(i = 0; i < 6; i++) printf("%lf", dvals[i]);
            printf("\n");

/* For accessing Product Specific Attributes in an HDF file */
            strcpy(sval," ");
            ret=PGS_MET_GetSetAttr(handles[INVENTORYMETADATA],
            "AdditionalAttributeName.1",&sval);

            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");
            strcpy(sval," ");
            "ParameterValue.1",&sval);
            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");
FORTRAN:

C For accessing Inventory Attributes in an HDF file

            dvals(1) = 0.0
            dvals(2) = 0.0
            dvals(3) = 0.0
            dvals(4) = 0.0
            dvals(5) = 0.0
            dvals(6) = 0.0

            ret = pgs_met_setattr_d(groups[INVENTORYMETADATA],
     1"GRingPointLatitude.1", dvals)
            print *, dvals(1), dvals(2), dvals(3), dvals(4),
     1 dvals(5), dvals(6)

C For accessing Product Specific Attributes in an HDF file

            sval = " "
            ret=pgs_met_setattr_s(groups[INVENTORYMETADATA],
     1"AdditionalAttributeName.1",sval)
            print *, sval
            sval = " "
            ret=pgs_met_setattr_s(groups[INVENTORYMETADATA],
     1"ParameterValue.1",sval)
            print *, sval
```

NOTES: (See notes 1,2,3, and 4 in PGS_MET_SetAttrib()

REQUIREMENTS: PGSTK-0290 PGSTK-380

Accesses Metadata Parameters in HDF Products or Independent ASCII Files

NAME: **PGS_MET_GetPCAttr()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_GetPCAttr(
PGSt_PC_Logical  fileId,
PGSt_integer     version,
          char *          hdfAttrName,
          char *          parmName,
          void *          parmValue)
```

FORTRAN:

```
include "PGS_MET_13.f"
include "PGS_MET.f"
include "PGS_SMF.h"

integer function pgs_getpcattr(fileId, version, hdfAttrName, parmName,
parmValue)

character*          fileId
integer             version
character*          hdfAttrName
character*          parmName
'user defined'      parmValue
```

DESCRIPTION: Metadata parameters held in HDF attributes or in a separate ASCII file can be read using this tool

INPUTS:

Table 6-22. PGS_MET_GetPCAttr Inputs

Name	Description	Units	Min	Max
fileId	product file id	none	variable	variable
version	product version number	none	1	variable
hdfAttrName	name of HDF attribute containing metadata	none	N/A	N/A
parmName	metadata parameter name	none	N/A	N/A

OUTPUTS:

Table 6-23. PGS_MET_GetPCAttr Outputs

Name	Description	Units	Min	Max
attrValue	value of attribute to be passed back to the user	none	N/A	N/A

RETURNS:

Table 6-24. PGS_MET_GetPCAttr Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_PCREAD_ERR	"Unable to obtain <filename or attribute filename> from the PC table" Most likely that <filename or attribute filename> is not defined in the PCF
PGSMET_E_FILETOODL_ERR	"Unable to convert <filename> into an ODL format" error returns from lower level routines should explain the problem
PGSMET_E_AGGREGATE_ERR	Unable to create ODL aggregate <aggregate name> It definitely means that ODL routine has failed to allocate enough memory
PGSMET_E_SYS_OPEN_ERR	Unable to open pc attribute file Usually if the file does not exist at the path given, check the name and path of the file
PGSMET_E_ODLTOVAL_ERR	Unable to convert attribute values from the ODL format error returns from lower level routines should explain the problem
PGSMET_E_NULL_PARAMETER	The requested parameter is a null value
PGSMET_E_NOT_SET	The requested parameter is not set

EXAMPLES:

```
C:
      char grpName[100];

/* For accessing Inventory Attributes in an HDF file */

            for(i = 0; i < 6; i++) dvals[i] = 0.0;
            ret = PGS_MET_GetPCAttr(MODIS_FILE, 1, "coremetadata",
            "GRingPointLatitude.1", dvals);
            for(i = 0; i < 6; i++) printf("%lf", dvals[i]);
            printf("\n");

/* For accessing Product Specific Attributes in an HDF file */

            strcpy(sval," ");
            ret=PGS_MET_GetPCAttr(MODIS_FILE,1,"coremetadata",
            "TestingAttribute1",&sval);
            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");

/* For accessing attributes in the ASCII Metadata file */
/* NOTE: For retrieving attribute values from the ASCII metadata file, users
have to generate a group name first before calling the function
```

```
PGS_MET_GetPCAttr. The procedures are as follows:
1:
      In this case the group name is INVENTORYMETADATA
      sprintf(grpName, "%s%s", PGSd_MET_GROUP_STR, "INVENTORYMETADATA");
2:
      ret = PGS_MET_GetPCAttr(10268, 1, grpName, "REPROCESSINGPLANNED",
      &sval);

*/

            strcpy(sval," ");
            sprintf(grpName, "%s%s", PGSd_MET_GROUP_STR,
      "INVENTORYMETADATA");
            ret = PGS_MET_GetPCAttr(10268, 1, grpName,
      "REPROCESSINGPLANNED",  &sval);
            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");

/* For LandSat7 Metadata output file */
/* NOTE: For retrieving the attribute from the Landsat7  meta file, users have
to generate a group name first before calling the function PGS_MET_GetPCAttr.
The procedures are as follows:

1:
      In this case the group name is "FORMAT_SUBINTERVAL_METADATA_1"
      sprintf(grpName,"%s%s",PGSd_MET_LSAT_GRP_STR,
      "FORMAT_SUBINTERVAL_METADATA_1");

2:
      ret = PGS_MET_GetPCAttr(10269, 1, grpName,
      "CONTACT_PERIOD_START_TIME", &sval);

*/
            strcpy(sval," ");
            sprintf(grpName,"%s%s",PGSd_MET_LSAT_GRP_STR,
      "FORMAT_SUBINTERVAL_METADATA_1");
            ret = PGS_MET_GetPCAttr(10269, 1, grpName,
      "CONTACT_PERIOD_START_TIME", &sval);
            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");

FORTRAN:

            char grpName[100];

C For accessing Inventory Attributes in HDF file

            for(i = 0; i < 6; i++) dvals(i) = 0.0
            ret = pgs_met_getpcattr_d(MODIS_FILE, 1, "coremetadata",
     1 "GRingPointLatitude.1", dvals)
            print *, dval(1), dval(2), dval(3), dval(4), dval(5),
     1 dval(6)

C For accessing Product Specific Attributes in HDF file

            sval = " "
            ret=pgs_met_getpcattr_s(MODIS_FILE, 1, "coremetadata",
     1 " TestingAttribute1",&sval)
            print *, sval

C For accessing attributes in ASCII Metadata file

            sval = " "
            ret = pgs_met_getpcattr_s(10268, 1, grpName,
     1 "REPROCESSINGPLANNED", &sval)
            print *, sval

C For Landsat7 Metadata file
            sval = " "
            grpName(1:)=PGSd_MET_LSAT_GRP_STR//
     1"FORMAT_SUBINTERVAL_METADATA_1"
            ret = pgs_met_getpcattr_s(10269, 1, grpName,
     1 "CONTACT_PERIOD_START_TIME", &sval
            print *, sval
```

NOTES: See Notes 1,2,3, and 4 in PGS_MET_SetAttr

In the ECS production environment all input files are accompanied by an ASCII version of the metadata (the .met file) so PGS_MET_GetPCAttr will always read metadata from the .met file. In the SCF environment if the data input file is in HDF a .met file need not be present and the metadata can be read from the file itself. This is an example of how an HDF input file should be designated in the PCF:

```
10253|hdfinputfile|/my/product/directory/|||hdfinputfile|1
```

The file names in the second and sixth fields must be identifal. If the input file is not in HDF, the metadata will be read from an ASCII file which must be separately identified in the sixth field of the input product entry of the PCF, as shown in this example:

```
10253|inputfile|/my/product/directory/|||inputfile.met|1
```

The .met file must have the same name as the product input file, with the .met extension appended. This file must be placed in the same directory as the input file.

Effective with the November 1996 SCF Toolkit delivery, the separate ASCII file can now be in the same format as the output from PGS_MET_Write().

In the ECS production environment the ASCII metadata file that accompanies a data input file delivered by Science Data Server does not contain archive metadata. For this reason, archive metadata can only be read from input files that are in HDF. If used to read a value for a metadata attribute that is contained in an HDF global text attribute named “archivemetadata” or “productmetadata” PGS_MET_GetPCAttr will attempt to read the metadata from the HDF file, even though an ASCII .met file is present. In all other cases, PGS_MET_GetPCAttr reads the ASCII .met file.

The ASCII file may be in one of two formats; either that written out by the PGS_MET_Write() routine or simple parameter=value construct. These formats are shown below for a simple case

```
OBJECT = SOMEPARAMETER

        NUM_VAL = 1

        VALUE = 200

END_OBJECT = SOMEPARAMETER
```

or

```
SOMEPARAMETER = 200
```

Note that if a parameter appears twice in the ASCII file (with the same parameter name and Class extension) only the first occurrence will be returned.

REQUIREMENTS: PGSTK-0290 PGSTK-0235

Accesses Configuration Data in the PC Table

NAME: **PGS_MET_GetConfigData()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_GetConfigData(
        char*           attrName,
        void*           attrValue)
```

FORTRAN:

```
include "PGS_MET_13.f"
include "PGS_MET.f"
include "PGS_SMF.h"

integer function pgs_met_getconfigdata( attrName, attrValue)
character*      attrName
'user defined'  attrValue
```

DESCRIPTION: Certain configuration parameters are held in the PC table as follows

10220|REMOTEHOST|sandcrab

This tool would retrieve the value "sandcrab" from the PC table given the name of the parameter "REMOTEHOST". The parameter id 10220 is not used here. The value string (e.g.. sandcrab) is assumed to be in ODL format and therefore different types are supported.

INPUTS:

Table 6-25. PGS_MET_GetConfigData Inputs

Name	Description	Units	Min	Max
attrName	name of parameter in PCF	none	N/A	N/A

OUTPUTS:

Table 6-26. PGS_MET_GetConfigData Outputs

Name	Description	Units	Min	Max
attrValue	value of attribute to be passed back to the user	none	N/A	N/A

RETURNS:

Table 6-27. PGS_MET_GetConfigData Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_AGGREGATE_ERR	"Unable to create ODL aggregate <aggregate name>" This should never occur unless the process runs out of memory
PGSMET_E_CONFIG_VAL_STR_ERR	"Unable to obtain the value of configuration parameter <name> from the PCF file". Likelihood is that either the parameter does not exist in the PCF or the PCF itself is in error which can be tested using pccheck.
PGSMET_E_CONFIG_CONV_ERR	"Unable to convert the value of configuration parameter <name> from the PCF file into an ODL format". Its most likely that the string values is not in ODL format.

EXAMPLES:

C:

```
/* These values must be defined in the PCF otherwise error is returned
*/
            ret = PGS_MET_GetConfigData("REV_NUMBER", &ival);
            strcpy(datetime, "");
            ret = PGS_MET_GetConfigData("LONGNAME", &datetime);
            dval = 0;
      ret = PGS_MET_GetConfigData("CENTRELATITUDE", &dval);
            printf("%d %lf %s\n", ival, dval, datetime);

FORTRAN:

C Retrieve some values from the PCF files. These must be
C defined in the PCF, otherwise the routine would return error
C Note the way _i for integer, _d for double and _s for strings are used
C at the end of the function name. This is necessary because fortran
C compiler would complain about type conflicts if a generic name
C is used
            ret = pgs_met_getconfigdata_i("REV_NUMBER", ival)
            datetime = ""
            ret = pgs_met_getconfigdata_s("LONGNAME", datetime)
            dval = 0
      ret = pgs_met_getconfigdata_d("CENTRELATITUDE", dval)
            if(ret.NE.PGS_S_SUCCESS) then
            print *, "GetConfigData failed.
            endif
```

```
        print *, ival, dval, datetime
```

NOTES: See Notes 1, 2, 3, and 4 for PGS_MET_SetAttr().

Although This tool ignores the first field in the PCF file depicting the config id, it is still important that this field is unique for the PC utility to function correctly User is responsible for the returned buffers to be large enough to hold the returned values.

Addendum for TK5.1

This routine now simply retrieves the values from the PCF and does not perform type and range checking. The user is still required to assign enough space for the returned values.

REQUIREMENTS: PGSTK-0290 PGSTK-0380

Write Metadata and their Values to HDF Attributes and/or ASCII Output Files

NAME: **PGS_MET_Write()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_Write(
        PGSt_MET_handle    mdHandle,
        char *             hdfAttrName,
        PGSt_integer       hdfFileId)
```

FORTRAN:

```
include 'PGS_MET_13.f'
include 'PGS_MET.f'
include 'PGS_SMF.h'

integer function pgs_met_write(mdHandle, hdfAttrName, hdfFileId)

        character*   mdHandle
        character*   hdfAttrName
        integer      hdfFileId
```

DESCRIPTION: This is the final tool that PGE uses when all the metadata parameters are set in memory. The tool checks that all the mandatory parameters are set.

INPUTS:

Table 6-28. PGS_MET_Write Inputs

Name	Description	Units	Min	Max
mdHandle	metadata group in MCF	none	N/A	N/A
hdfAttrName	HDF attribute name to contain metadata	none	N/A	N/A
hdfFileId	HDF file ID	none	N/A	N/A

OUTPUTS: None

RETURNS:

Table 6-29. PGS_MET_WriteReturns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_NO_INITIALIZATION	Metadata file is not initialized
PGSMET_E_ODL_MEM_ALLOC	ODL routine failed to malloc memory space
PGSMET_E_GROUP_NOT_FOUND	No group called <name> found in the MCF
PGSMET_E_OPEN_ERR	Unable to open <temporary> file with file id <fileId>
PGSMET_E_SD_SETATTR	Unable to set the HDF file attribute. Note: HDF4.0r2 and previous versions of HDF have imposed a limit.
PGSMET_E_MALLOC_ERR	Unable to allocate memory for the hdf attribute
PGSMET_E_MAND_NOT_SET	Some of the mandatory parameters were not set
PGSMET_E_FGDC_ERR	Note: HDF attribute is still written out. Unable to convert UTC input date time string to FGDC values
PGSMET_E_ILLEGAL_HANDLE	Handle is illegal. Check that initialization has taken place.
PGSMET_E_HDFFILENAME_ERR	Unable to obtain HDF filename.
PGSMET_E_ASCII_ERR	Unable to open MET ASCII file.

EXAMPLES:

C:

```
/* Write to ASCII metadata file for non-HDF output product */
            ret= PGS_MET_Write(handles[ODL_IN_MEMMORY],NULL, 101);
            if(ret != PGS_S_SUCCESS)
                  {
                  printf("ASCII Write failed\n");
                  }
/* Write to HDF file */
            ret= PGS_MET_Write(handles[INVENTORYMETADATA],  "metadata", sdid);
            if(ret != PGS_S_SUCCESS)
                   {
                   printf("HDFWrite failed\n");
                   }
```

FORTRAN:

```
C Write to ASCII file for non-HDF output product
                  result= pgs_met_write(groups(ODL_IN_MEMORY),dummyStr, 101)
                  if(result.NE.PGS_S_SUCCESS.AND. result.NE.PGSMET_MAND_NOT_SET)
                  then
     1            print *,"ASCII Write failed"
                  endif
```

```
C Write to HDF file
                  result= pgs_met_write(groups(INVENTORYMETADATA),
     1            "coremetadata", sdid)
                  if(result.NE.PGS_S_SUCCESS.AND. result.NE.PGSMET_MAND_NOT_SET)
                  then
     1            print *,"ASCII Write failed"
                  endif
```

NOTES: When writing an attribute which has been defined as "UNSIGNED INT", the value written to the ASCII or HDF file may appear negative. The user should use the type “unsigned int” or the ECS equivalent (PGSd_uinteger) to interpret the value correctly. (see Note 4 of PGS_MET_SetAttr in Section 6.2.1.4.)

This routine can be used multiple times to write/attach separate master groups as local or global HDF attributes. To attach a mastergroup to a local element in an HDF file, an sds_id must be passed in as an argument, rather than an sd_id(hdfFileId). **!!!NOTE!!!** : Attaching metadata to a local element using the Toolkit is not standard practice for HDF-EOS files and should be avoided.

When writing the inventory metadata (MASTERGROUP = INVENTORYMETADATA in the MCF, mdHandle = coremetadata in the function call) to an HDF file, an ASCII version of the metadata is automatically created in the data product output directory. It is given the same name as the data product output, with the extension .met, i.e. ProductName.met. If the data product output is not in HDF, the following lines must be included in the PCF in order to create this required .met file:

```
?PRODUCT OUTPUT

100|ProductName|my/output/directory|||1
.
.
.
? USER RUNTIME PARAMETERS

101|ProductMetadataFile|100:1
```

where the second field is simply a comment.

An ASCII version of the metadata file will be created in the execution directory with the name *ProductName.met*. The user needs to call PGS_MET_Write with mdHandle[0], the HDF attribute name set to NULL and the identifier set to the logical identifier in the PCF.

2. If MANDATORY parameters are not set, an error PGSMET_E_MAND_NOT_SET is returned only in a PGE. The value of the metadata is set to as follows:

DATA_LOCATION	VALUE
PGE	"NOT SET"

PCF	"NOT FOUND"
MCF	"NOT SUPPLIED"
TK	“NOT OBTAINED”
DSS	“NOT PROVIDED”
DAAC	“NOTSUPPORTED”
DP	“NOT INCLUDED”

The writing of the hdf header is not affected

NOTE: A warning PGSMET_W_METADATA_NOT_SET is issued if MANDATORY has the value FALSE in the MCF, and the specific attribute will not appear in the HDF-EOS attribute or the ASCII file.

3. Only system errors such as memory failure, file openings etc. should be able to abort the write procedure.

4. NUM_VAL and CLASS fields are written in the HDF header

For metadata of type DATETIME, additional metadata is produced:

CALENDATDATETIME becomes CALENDARDATE and TIMEOFDAY.

RANGEBEGININGDATETIME becomes RANGEBEGININGDATE and RANGEBEGININGTIME

RANGEENDINGDATETIME becomes RANGEENDINGDATE and RANGEENDINGTIME

The user no longer has to worry about the size of the MCF exceeding the HDF limit on attribute sizes. This is now handled internally. The user simply needs to set coremetadata (or archivemetadata) and if the limit is exceeded, coremetadata.0, .1, etc. are produced.

REQUIREMENTS: PGSTK-0290, PGSTK-0380, PGSTK-0400, PGSTK-0450, PGSTK-0510

Free Memory of MCFs

NAME: **PGS_MET_Remove()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_Remove()
```

FORTRAN:

```
include "PGS_MET_13.f"
include "PGS_MET.f"
include "PGS_SMF.h"

integer function pgs_met_remove()
```

DESCRIPTION: This routine removes ODL representation of all MCF files and some internal files used by the MET tools.

INPUTS: None

OUTPUTS: None

RETURNS: None

EXAMPLES:

C:

```
result = PGS_MET_Remove();
printf("SUCCESS\n");
return 0;
```

FORTRAN:

```
print *, ival, dval, datetime
result = pgs_met_remove()
print *, "SUCCESS"
end
```

NOTES: This routine must be called by the user before the program terminates.

REQUIREMENTS: PGSTK-0430

6.2.1.5 Data Quality Assurance

The tools in this section will be used to support the analysis of Q/A data output from the production processes. There is no Toolkit tool to meet this requirement, however, this requirement is being met by other HDF functionality

REQUIREMENTS: PGSTK-0510

6.2.1.6 Temporary and Intermediate Files

This section contains descriptions of tools that are specific to temporary and intermediate file I/O. A temporary file is a file that exists only for the duration of a single PGE; it is deleted following successful PGE termination. An intermediate file exists for a user-defined time after the PGE terminates.

After you open a temporary or intermediate file, use the native C or FORTRAN I/O routines to perform I/O.

Note that there are no "Temp_Close" tools; use the Gen_Close tools to close files. See "Generic File I/O Tools" (Section 6.2.1.3).

Special note regarding FORTRAN 90: Tools PGS_IO_Gen_OpenF and PGS_IO_Gen_Temp_OpenF now have FORTRAN 90 versions. These versions support two specific usages of the F90 OPEN function that are not supported in ANSI FORTRAN 77; they do not support all F90 options of OPEN. At Toolkit installation time, you select between F77 and F90, and the appropriate source code file is compiled; the function names are the same in both versions of FORTRAN. Options and text that apply only to FORTRAN 90 are marked in this document as ***F90 SPECIFIC***.

IMPORTANT CHANGES FROM TOOLKIT 4

The following environment variables MUST be set to assure proper operation:

PGS_PC_INFO_FILE	path to process control file

However, the following environment variables are NO LONGER recognized by the Toolkit:

PGS_TEMPORARY_IO	path to temporary files
PGS_INTERMEDIATE_INPUT	path to intermediate input files
PGS_INTERMEDIATE_OUTPUT	path to intermediate output files

Instead, the default paths, which were defined by these environment variables in previous Toolkit releases, may now be specified as part of the Process Control File (PCF). Essentially, each has been replaced by a global path statement for each of the respective subject fields within the PCF. To define a global path statement, simply create a record that begins with the '!' symbol defined in the first column, followed by the global path to be applied to each of the records within that subject field. Only one such statement can be defined per subject field and it must appear prior to any dependent subject entry.

The status condition PGSIO_E_GEN_BAD_ENVIRONMENT now indicates an error status on the global path statement as defined in the PCF, and NOT on an environment variable. However, as with previous releases, the status message associated with this condition may reference the above “tokens,” but this is only to indicate which of the global path statements is problematic.

The following environment variable is also NO LONGER recognized.

PGS_HOST_PATH	path to internet hosts file (e.g., /etc/hosts)

The temporary I/O tools will look to the Process Control File (PCF) for the runtime parameter PGSd_IO_Gen_HostAddress to obtain the IP address.

Open a Temporary/Intermediate File (C Version)

NAME: **PGS_IO_Gen_Temp_Open( )**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_Gen_Temp_Open(
        PGSt_IO_Gen_Duration       file_duration,
        PGSt_PC_Logical            file_logical,
        PGSt_IO_Gen_AccessType     file_access,
        PGSt_IO_Gen_FileHandle**   file_handle);
```

FORTRAN: (not applicable)

DESCRIPTION: This routine lets the user create and open Temporary and Intermediate files with a variety of access modes. The returned argument PGSt_IO_Gen_FileHandle is directly compatible with the standard "C" library stream I/O manipulation routines.

INPUTS: file_duration:

```
        PGSd_IO_Gen_Endurance          // Creates Intermediate File //
        PGSd_IO_Gen_NoEndurance        // Creates Temporary File  //
```

file_logical-User defined logical file identifier

file_access-type of access granted to opened file:

Table 6-30. File Access Type

Toolkit	C	Description
PGSd_IO_Gen_Read	"r"	Open file for reading
PGSd_IO_Gen_Write	"w"	Open file for writing, truncating existing file to 0 length, or creating a new file
PGSd_IO_Gen_Append	"a"	Open file for writing, appending to the end of existing file, or creating file
PGSd_IO_Gen_Update	"r+"	Open file for reading and writing
PGSd_IO_Gen_Append Update	"a+"	Open file for reading and writing, to the end of existing file, or creating a new file; whole file can be read, but writing only appended

OUTPUTS: file_handle-used to manipulate files with other "C" library stream I/O routines

RETURNS:

Table 6-31. PGS_IO_Gen_Temp_Open Returns

Return	Description
PGS_S_SUCCESS	Success
PGSIO_W_GEN_ACCESS_MODIFIED	Illegal attempt to open existing file for access mode PGSd_IO_Gen_Write or PGSd_IO_Gen_Trunc; Access mode reset to PGSd_IO_Gen_AppendUpdate
PGSIO_W_GEN_NEW_FILE	File expected, but was missing; new file created
PGSIO_W_GEN_DURATION_NOMOD	Attempt to alter existing intermediate duration attribute ignored
PGS_E_UNIX	UNIX system error
PGSIO_E_GEN_OPENMODE	Invalid access mode
PGSIO_E_GEN_REFERENCE_FAILURE	Other error accessing $PGS_PC_INFO_FILE
PGSIO_E_GEN_BAD_FILE_DURATION	Invalid file duration
PGSIO_E_GEN_FILE_NOEXIST	No entry for file logical in $PGS_PC_INFO_FILE
PGSIO_E_GEN_CREATE_FAILURE	Error creating new file entry in $PGS_PC_INFO_FILE
PGSIO_E_GEN_NO_TEMP_NAME	Failed to create temporary filename
PGSIO_E_GEN_BAD_ENVIRONMENT	Bad environment detected for I/O path **...**

“Existing file” means that an entry for the file exists in $PGS_PC_INFO_FILE.

(NOTE: the above are short descriptions only; full text of messages appears in files $PGSMSG/*.t . Descriptions may change in future releases depending on external ECS design.)

EXAMPLE:

```
// This example illustrates how to create an Intermediate
   File //

PGSt_SMF_status         returnStatus;
PGSt_PC_Logical         logical;
PGSt_IO_Gen_FileHandle  *handle;

#define     INTER_1B 101

returnStatus =
PGS_IO_Gen_Temp_Open(PGSd_IO_Gen_Endurance,INTER_1B,
                     PGSd_IO_Gen_Write,&handle );
if (returnStatus != PGS_S_SUCCESS)
{
    goto EXCEPTION;
}
    .
    .
    .
EXCEPTION:
```

NOTES: This function will support most POSIX modes of fopen; the only exception being truncate mode (w+).

Logical identifiers used for files may NOT be duplicated.

Existing files will NOT be overwritten by calling this function in mode PGSd_IO_Gen_Write. Instead, they will be opened in PGSd_IO_Gen_AppendUpdate mode; a warning will be issued signifying that this is the case. Warnings will also be issued in the event that a non-existent file is opened in modes other than explicit write (i.e., PGSd_IO_Gen_Append, or PGSd_IO_Gen_AppendUpdate).

By using this tool, the user understands that a Temporary file may only exist for the duration of a PGE. Whether or not the user deletes this Temporary file prior to PGE termination, it will be purged by the Science Data Processing Segment (SDPS) system during normal cleanup operations. If the user requires a more static instance of a file, one that will exist beyond normal PGE termination, that user may elect to create an Intermediate file instead by specifying some persistence value (currently, PGSd_IO_Gen_Endurance is the only value recognized); note that this value is only valid for the initial creation of a file and will not be applied to subsequent accesses of the same file.

The following table gives proper use of the *file_duration* input variable:

Table 6-32. Proper Use of Persistence Values

File Type & Access	Duration Factors
TEMPORARY	
Creation	PGSd_IO_Gen_NoEndurance
Repeated Access	NULL
INTERMEDIATE	
Creation	PGSd_IO_Gen_Endurance
Repeated Access	NULL

FILE CHARACTERISTICS

All files created by this function have names of the following form:

[label][production-run-id][local-network-IP-address][process-id][date][time]

as in:

```
'pcrrrrrrrvvvwwxxppppppdddyyhhmmss'
```

Table 6-33. Temporary File Name Definition

Field	Description	Format
label	SDP Toolkit Process Control	“pc”
production-run-id	numeric identifier from 1 to 8 places	rrrrrrrr
local-network-IP-address	local portion of Internet protocol (IP) address uuu.vvv.ww.xx	vvvwwxx
process-id	UNIX identifier for current process	pppppp
date	# days from beginning of year, and the year	dddyy
time	time from midnight local time	hhmmss

```
ex.       pc115028320102800039510 4034

          pc 1   1502832 010280 00395 104034
label_____|  |   |       |      |     |
prod.-run-id_|   |       |      |     |
local-IP-address_|       |      |     |
process-id_______________|      |     |
date____________________________|     |
time__________________________________|
```

All temporary and intermediate files generated by this tool are unique within a certain network locale (e.g., DAAC). Also, all file names are a maximum of 34 characters in length.

NOTE: Users should NOT put entries in the TEMP or INTERMEDIATE OUTPUT sections. The Toolkit will do this.

REQUIREMENTS: PGSTK-0530, PGSTK-0531

Open a Temporary/Intermediate File (FORTRAN Version)

NAME: **PGS_IO_Gen_Temp_OpenF( )**

SYNOPSIS:

C: (not applicable)

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_IO.f'
INCLUDE     'PGS_IO_1.f'

integer function pgs_io_gen_temp_openf(file_duration, file_logical,
                            file_access, record_length, file_handle)
integer file_duration
integer file_logical
integer file_access
integer record_length
integer file_handle
```

DESCRIPTION: Upon a successful call, this function will return a logical unit number for use with FORTRAN READ and WRITE statements. This is returned to the user via the parameter file_handle. The user provides the logical file identifier that internally gets mapped to the associated physical file. The user also provides the file duration parameter, to specify whether the file being opened is to be temporary or intermediate.

INPUTS: file_duration-specifies how long file will last:

Table 6-34. File Duration

PGS-defined value	Description
PGSd_IO_Gen_Endurance	intermediate file
PGSd_IO_Gen_NoEndurance	temporary file

file_logical-User defined logical file identifier

file_access-type of access granted to opened file:

Table 6-35. File Access Type

PGS FORTRAN Access Mode	Rd/Wr/Update/Append	FORTRAN 77/90 'access='	FORTRAN 77/90 'form='
PGSd_IO_Gen_RSeqFrm	Read	Sequential	Formatted
PGSd_IO_Gen_RSeqUnf	Read	Sequential	Unformatted
PGSd_IO_Gen_RDirFrm	Read	Direct	Formatted
PGSd_IO_Gen_RDirUnf	Read	Direct	Unformatted
PGSd_IO_Gen_WSeqFrm	Write	Sequential	Formatted
PGSd_IO_Gen_WSeqUnf	Write	Sequential	Unformatted
PGSd_IO_Gen_WDirFrm	Write	Direct	Formatted
PGSd_IO_Gen_WDirUnf	Write	Direct	Unformatted
PGSd_IO_Gen_USeqFrm	Update	Sequential	Formatted
PGSd_IO_Gen_USeqUnf	Update	Sequential	Unformatted
PGSd_IO_Gen_UDirFrm	Update	Direct	Formatted
PGSd_IO_Gen_UDirUnf	Update	Direct	Unformatted
F90 SPECIFIC			
PGSd_IO_Gen_ASeqFrm	Append	Sequential	Formatted
PGSd_IO_Gen_ASeqUnf	Append	Sequential	Unformatted

record_length-record length for direct access IO:
mandatory for direct access (minimum value = 1)
ignored otherwise

F90 SPECIFIC must be greater than or equal to 0 for sequential access; if 0, file is opened with default record length

OUTPUTS: file_handle-used to manipulate files with READ and WRITE

RETURNS:

Table 6-36. PGS_IO_Gen_Temp_OpenF Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_NO_FREE_LUN	All logical unit numbers are in use
PGSIO_W_GEN_ACCESS_MODIFIED	The access mode has been modified
PGSIO_E_GEN_OPENMODE	Illegal open mode was specified
PGSIO_E_GEN_OPEN_OLD	Attempt to open with STATUS=OLD failed
PGSIO_E_GEN_OPEN_NEW	Attempt to open with STATUS=NEW failed
PGSIO_E_GEN_OPEN_RECL	Invalid record length specified
PGSIO_W_GEN_OLD_FILE	File exists: changing access to update
PGSIO_W_GEN_NEW_FILE	File not found, created new one
PGSIO_W_GEN_DURATION_NOMOD	Illegal attempt to modify file duration
PGSIO_E_GEN_REFERENCE_FAILURE	Can't do Temporary file reference
PGSIO_E_GEN_BAD_FILE_DURATION	Illegal file duration value
PGSIO_E_GEN_FILE_NOEXIST	File not found, cannot create
PGSIO_E_GEN_CREATE_FAILURE	Unable to create new file
PGSIO_E_GEN_NO_TEMP_NAME	New name could not be generated

EXAMPLE:

```
integer returnstatus
integer file_duration
integer file_logical
integer file_access
integer record_length
integer file_handle

file_duration     = PGSd_IO_Gen_NoEndurance
file_logical= 101
file_access       = PGSd_IO_Gen_WDirUnf
record_length     = 1

returnstatus = PGS_IO_Gen_Temp_OpenF(file_duration,
                                     file_logical,
                                     file_access,
                                     record_length,
                                     file_handle)
```

```
if (returnstatus .NE. PGS_S_SUCCESS) then
                 C   goto 1000
                  endif
                  .
                  .
                  .
100 <error handling goes here>
```

NOTES: Logical identifiers used for Temporary and Intermediate files may NOT be duplicated. Existing files will NOT be overwritten by calling this function in any of the write modes. Instead, they will be opened in the corresponding update mode; a warning will be issued signifying that this is the case. Warnings will also be issued in the event that a nonexistent file is opened in modes other than explicit write.

By using this tool, the user understands that a Temporary file may only exist for the duration of a PGE. Whether or not the user deletes this file prior to PGE termination, it will be purged by the PGS system during normal cleanup operations. If the user requires a more static instance of a file, one that will exist beyond normal PGE termination, that user may elect to create an Intermediate file instead by specifying some persistence value (currently, PGSd_IO_Gen_Endurance is the only value recognized); note that this value is only valid for the initial creation of a file and will not be applied to subsequent accesses of the same file.

In order to insure that generated temporary file names are unique for the same host, a delay factor of 1 millisecond is imposed during the name creation process.

Due to the nature of FORTRAN IO, it is possible to write a file opened for reading as well as read a file opened for writing. The matching of access mode to IO statement cannot be enforced by the tool. This is up to the user.

Once a file has been opened with this tool, it must be closed with a call to PGS_IO_Gen_CloseF before being re-opened. Failure to do this will result in undefined behavior.

REQUIREMENTS: PGSTK-0530, PGSTK-0531

Delete a Temporary File

NAME: **PGS_IO_Gen_Temp_Delete( )**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_Gen_Temp_Delete(
        PGSt_PC_Logical     file_logical);
```

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_IO_1.f'

integer pgs_io_gen_temp_delete(
        integer file_logical)
```

DESCRIPTION: Upon a successful call, this function will “effectively” delete the Temporary file currently referenced by the specified logical identifier. (See NOTES.) Future references to this logical identifier will no longer provide access to a file until such time as a new temporary file is created with the same logical identifier.

INPUTS: file_logical-User defined logical file identifier

OUTPUTS: None

RETURNS:

PGS_S_SUCCESS
PGSIO_E_GEN_REFERENCE_FAILURE
PGSIO_E_GEN_FILE_NODEL
PGSIO_W_GEN_FILE_NOT_FOUND

EXAMPLE:

```
PGSt_SMF_status   ret_val;
PGSt_PC_Logical   logical;

#define     INTER_1B 101

ret_val = PGS_IO_Gen_Temp_Delete( INTER_1B );
if (ret_val != PGS_S_SUCCESS)
{
    goto EXCEPTION;
}
    .
    .
    .
EXCEPTION:
```

NOTES: The actual deletion of Temporary files is not carried-out until after the completion of the PGE run. Instead, these files are marked as deleted through the Process Control mechanism. This allows for the preservation of all Temporary files generated during a PGE run, to facilitate error tracking/debugging following a failed run of a PGE. This in no way prevents the creation of a new temporary file using the same logical identifier as one previously deleted.

Unlike all other IO_Gen tools, this function has a FORTRAN binding to C. There is no separate FORTRAN version.

Logical identifiers used for Temporary and Intermediate files may NOT be duplicated.

By using this tool, the user understands that a truly Temporary file may only exist for the duration of a PGE. Whether or not the user deletes this file prior to PGE termination, it will be purged by the Science Data Processing System (SDPS) system during normal cleanup operations.

REQUIREMENTS: PGSTK-0520

6.2.2 Error/Status Reporting (SMF Tools)

To detect and report on error and status conditions in a consistent manner across the ECS, standardized status messages and status codes must first be established. The method used to institute these message/code pairs is by way of the 'smfcompile' utility. But first, users will need to create Status Message Files (SMFs) to contain their custom status messages and corresponding status identifiers. These identifiers take the form of user defined mnemonics that visually convey the essence of the status message. The user will make direct use of these mnemonics in their software when testing for status conditions and when interfacing with the SMF Toolkit functions. Once an SMF is completed, the smfcompile utility is run in order to bind the status messages and mnemonics with integral status codes. This process facilitates the runtime access of all status messages and provides for the referencing of status mnemonics within the user's code.

The status codes generated by the 'smfcompile' utility are guaranteed to be unique across the entire SDPS system to ensure that there will be no ambiguous status conditions, in the event that code from different Science Computing Facilities (SCFs) is merged into a single executable and/or PGE. This uniqueness is possible because "seed" values, which are different for every SMF, are used in the generation of the status codes. Typically, many SMF files will be created in the course of software development; therefore many seed numbers will be required. <u>However, it is important to note that valid seed numbers can only be obtained from the Toolkit development team (pgstlkit@eos.hitc.com). Any attempt to produce SMFs from "home-grown" seed values may result in the SMFs being unusable at integration & test time.</u>

The SDP Toolkit routines actually contain their own collection of status codes and associated status messages for describing the state of each Toolkit function. Users of the Toolkit functions should examine the return values of each tool before performing any other action. To inform a calling unit (user's software) about the exit state of a called Toolkit routine, each Toolkit function sets a status message and assigns a status code to the return value. On the basis of its interpretation of this return value, the calling unit may elect to perform some error handling. As part of this procedure, the user should either propagate the existing status code up through their calling hierarchy, or set a status code and message to represent the outcome of any local error handling attempt.

Upon detection of an error state, users are advised to report on the existing error prior to performing an error handling procedure. The content of these reports might include the following: a user-defined message string to convey the nature of the status condition, a user-defined action string to indicate the next operation to be performed in response to the status condition, and a system defined string that uniquely identifies the environment in which the status condition occurred. However, this is merely a suggestion; the user is free to define the content of the status reports to satisfy their own requirements. The method for reporting this information will involve the generation of a report from the information just described and the subsequent transmission of that report to the appropriate destination(s).

Once software development has been completed, all the Status Message Files (SMFs) created to support that development will be delivered to the DAAC along with the developed PGE(s). The

Toolkit SMFs will be delivered to the DAACs along with the Toolkit library, just as they were delivered to the SCFs.

The tools provided here allow for the propagation of status information within a PGE executable to facilitate a user's error handling process. They also provide the means to communicate status and error information to various monitoring authorities and event logs. Additionally, there is a tool that enables the user to specify, a priori, the action to be taken in the wake of a fatal arithmetic event. This mechanism will allow the user to take their own corrective measures to control an event that is terminal by default. Note that all other event conditions fall under the purview of system processing and are thereby controlled by the governing SDPS software.

Several new features have been incorporated into these tools for Toolkit 5 in order to improve their efficiency. One of those features allows for the buffering of individual status messages up to some user defined runtime limit. This should greatly reduce the amount of I/O required to access these messages. As a process proceeds to completion, new status messages are buffered as older, less used status messages become unbuffered. The goal here is to only access status messages from their runtime file when they are being referenced for the first time. The actual observed improvement will depend on the degree to which a process' status messages are localized (i.e., A particular status message should ideally only be referenced within a short body of code.) and the buffer size, which is set by the user. Another feature reduces the number of replicated status messages that can appear in the status log file. This is accomplished by "compressing" duplicate messages into a count of such messages. This feature should significantly reduce the size of the status log file and contribute to its general readability.

Please refer to Appendix B for guidance on the creation of Status Message Files and for examples of SMFs and explicit SMF Toolkit usage.

6.2.2.1 Log File Output Control

Several new features have been added to the Toolkit to allow greater control of message logging. The behavior of these features is controlled via entries in the Process Control File (PCF). Note that the use of some or all of these features may be strictly controlled at the DAACs.

6.2.2.1.1 Logging Control

PCF entry:
10114|Logging Control; 0=disable logging, 1=enable logging|1

This may be used to disable logging altogether. If logging is disabled NO message will output to any log files (although a small header will still be written to the log files indicating that for this PGE logging has been disabled). The default state is for logging to be enabled.

6.2.2.1.2 Trace Control

PCF entry:
10115|Trace Control; 0=no trace, 1=error trace, 2=full trace|0

This may be used to specify the trace level for message logging. Tracing is a feature made possible by the addition of two new SMF tools: PGS_SMF_Begin and PGS_SMF_End (see the respective entries in 6.2.2.2 Status Reporting Tools). Users may include these tools at the beginning and ending of their functions (respectively) to signal to the SMF system when each user defined function is entered and exited. Three levels of tracing are possible:

No Tracing
This is the default state. No information concerning the entering or exiting of functions is recorded to the log files. No information concerning the path of a function call is recorded to the log files.

Example Log Entry:

```
func4():PGSTD_W_PRED_LEAPS:27652
predicted value of TAI-UTC used (actual value unavailable)
```

Error Tracing
If error tracing is enabled, information concerning the path of a function call is recorded to the log files any time a status message is logged to a log file. This is useful in determining where in a chain of function calls an error occurred. No information concerning the entering or exiting of functions is recorded in this state.

Example Log Entry:

```
main():
   func1():
      func2():
         func3():
            func4():PGSTD_W_PRED_LEAPS:27652
            predicted value of TAI-UTC used (actual value unavailable)
```

Full Tracing
If full tracing is enabled, a message will be written to the log files each time a function is entered and exited (only those user functions with the PGS_SMF_Begin/End calls, see above). Indenting will also be done to show the path of each function call.

Example Log Entry:

```
PGS_SMF_Begin: main()

  PGS_SMF_Begin: func1()

   PGS_SMF_Begin: func2()

     PGS_SMF_Begin: func3()

      PGS_SMF_Begin: func4()

        func4():PGSTD_W_PRED_LEAPS:27652
        predicted value of TAI-UTC used (actual value unavailable)

      PGS_SMF_End: func4()
```

```
    PGS_SMF_End: func3()
PGS_SMF_End: func2()
PGS_SMF_End: func1()
PGS_SMF_End: main()
```

6.2.2.1.3 Process ID Logging

PCF entry:
10116|Process ID logging; 0=don’t log PID, 1=log PID|0

This may be used to enable the tagging of log file entries with the process ID of the process from which the entry came. This is useful for PGEs that run concurrent processes which will all be writing to a single log file simultaneously. If process ID logging is enabled, each log entry will be tagged with the process ID of the process making the entry. This can facilitate in post-processing a log file.

Example Log Entry:
```
func4():PGSTD_W_PRED_LEAPS:27652 (PID=2710)
predicted value of TAI-UTC used (actual value unavailable)
```

6.2.2.1.4 Status Level Control

PCF entry:
10117|Disabled status level list (e.g., W S F)|<status level list>

This may be used to disable the logging of status codes of specific severity levels. A list of levels to be disabled should be substituted for <status level list> (e.g.: N M U). No message of a status level indicated in the list will be recorded to any log file (see Appendix B for details on status message levels). The default state is to enable logging for all status levels.

6.2.2.1.5 Status Message Seed Control

PCF entry:
10118|Disabled seed list|<status code seed list>

This may be used to disable the logging of status codes generated from specific seed values. A list of seed values, the status codes derived from which should be disabled, should be substituted for <status code seed list> (e.g.: 3 5). No message derived from a seed value indicated in the list will be recorded to any log file (see Appendix B for details on status message seed values). The default state is to enable logging for all seed values.

6.2.2.1.6 Individual Status Code Control

PCF entry:
10119|Disabled status code list|<status code list>

This may be used to disable the logging of specific status codes. A list of status code mnemonics and/or numeric status codes should be substituted for <status code list> (e.g.: PGSTD_M_ASCII_TIME_FMT_B 678954). Note that only Toolkit status codes can be disabled by using mnemonics. To disable a user generated status code a numeric status code must be used. No messages whose status codes or mnemonics are included in the list will be recorded to any log file. The default state is to enable logging for all status codes.

6.2.2.1.7 Generating Runtime E-Mail Messages

A PGE may be configured to automatically generate and send e-mail message during runtime when specific user defined status codes are logged. This is done by assigning an e-mail action to a given user defined status code.

An e-mail action is an SMF code with the special status level of "C" and a mnemonic that begins with the characters "PGSEMAIL" (the rest of the mnemonic may contain any other valid mnemonic characters), for example:

PGS_*C_PGSEMAIL*_SEND_EMAIL
ASTER_*C_PGSEMAIL*_ALERT
MODIS_*C_PGSEMAIL*_ERROR

An e-mail message will be generated anytime a user defined status code with an associated e-mail action is logged via the SMF logging routines. The contents (body) of these messages will be the text (message) associated with the user defined status code. The subject of these messages will be the mnemonic associated with the user defined status code. The list of recipients is defined in the e-mail action definition.

Example:
In a user defined status message file the following status code mnemonic label and e-mail action mnemonic label have been defined (the e-mail action is associated with the status code via the "::" syntax):

```
MODIS_E_PGE_INIT_FAILED   The PGE failed to initialize.
                          ::MODIS_C_PGSEMAIL_NOTIFY
MODIS_C_PGSEMAIL_NOTIFY   john@modis.org, sue@modis.org
```

The following lines appear in a C source code file:

```
    returnStatus = initializePGE();
    if (returnStatus == MODIS_E_PGE_INIT_FAILED)
    {
        PGS_SMF_SetStaticMsg(returnStatus, "main()");
        exit(1);
    }
```

At runtime, if the returned status code from the function initializePGE() has the value defined by MODIS_E_PGE_INIT_FAILED, this status is logged via the SMF function

PGS_SMF_SetStaticMsg(), and because this status code has an e-mail action associated with it, an e-mail message will be generated.

The e-mail message will be sent to: sue@modis.org and john@modis.org
The subject field of the e-mail message will be: MODIS_E_PGE_INIT_FAILED
The text of the e-mail message will be: The PGE failed to initialize.

Note:
This functionality will be disabled at the DAACs.

6.2.2.2 Status Reporting Tools

Get Toolkit Version

NAME:	**PGS_SMF_GetToolkitVersion()**
SYNOPSIS:	

C:	#include <PGS_SMF.h>
	void
	PGS_SMF_GetToolkitVersion(
	char version[21]);

FORTRAN:	include‘PGS_SMF.f’ integer function pgs_smf_gettoolkitversion(character*20 version)
DESCRIPTION:	This function returns a string describing the current version of the Toolkit.
INPUTS:	None
OUTPUTS:	version - character string describing the current version of the Toolkit
RETURNS:	None
EXAMPLES:	

C:	char version[21];
	PGS_SMF_GetToolkitVersion(version);

FORTRAN:	character*20 call pgs_smf_gettoolkitversion(version)
NOTES:	User must allocate enough memory to hold the Toolkit version string.This function does not allocate any memory for the user.
REQUIREMENTS:	

Set UNIX Status Message

NAME: **PGS_SMF_SetUNIXMsg( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_SetUNIXMsg(
        PGSt_integer  unix_errcode,
        char          *msg,
        char          *funcname);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_setunixmsg(unix_errcode,msg,funcname)
        integer        unix_errcode
        character*240  msg
        character*32          funcname
```

DESCRIPTION: This tool provides the means to retain UNIX error messages for later retrieval. Additionally, the user has the flexibility to append a user defined message to a UNIX message for further clarity.

INPUTS: unix_errcode-the error code set by C library; UNIX system calls; and POSIX FORTRAN calls, i.e., the value stored in C 'errno' and Fortune 'IERROR'

msg-user defined status message string

funcname-function where the status condition occurred

OUTPUTS: None

RETURNS:

Table 6-37. PGS_SMF_SetUNIXMsg Returns

Return	Description
PGS_S_SUCCESS	Success
PGSSMF_E_LOGFILE	Error opening status, report or user files
PGSSMF_E_UNDEFINED_UNIXERRNO	Undefined UNIX error
PGSSMF_E_MSG_TOOLONG	Message length exceeded

EXAMPLES:

C: This example uses the ‘popen()’ C library routine merely to illustrate how the SMF tool PGS_SMF_SetUNIXMsg() might be used to preserve the UNIX error condition. Note that ‘popen()’ is not part of the POSIX standard and therefore should not be used within the science software.

```
PGSt_SMF_status Get_Listing()
{
   FILE                 *stream;
   char                 buffer[101];
   char                 directoryEntry[101];
   PGSt_SMF_statusreturnStatus = PGS_S_SUCCESS;

   if (stream = popen(“ls”,”r”) != NULL)
   {
      while (fgets(buffer,100,stream) != NULL)
         {
            scanf(buffer,”%s”,directoryEntry);
         }
   }
   else
   {
       PGS_SMF_SetUNIXMsg(errno,NULL,”Get_Listing()”);
       pclose(stream);
       returnStatus = PGS_E_UNIX;
   }
}
```

FORTRAN:

```
implicit none

integer     pgs_smf_setunixmsg
character*1 chr
integer     ierror

PXFFGETC(IPXFCONST(“STDIN_UNIT”),chr,ierror)
IF (ierror .NE. 0) THEN
   pgs_smf_Setunixmsg(ierror,’PXFFGETC() error
occured’,’Get_Listing()’)
ENDIF
```

NOTES: The parameter “funcname” can be passed in as NULL if you do not wish to record the routine that noted this error. However, it is strongly recommended that you pass the routine name for tracking purposes. Likewise, the parameter “msg” can be NULL unless you wish to have an additional message appended to the system defined UNIX message. The static variable ‘errno’ has been declared in ‘PGS_SMF.h’. Since UNIX treats errno as a static parameter, the user will have to save the value returned from the critical call unless the call to ‘PGS_SMF_SetUNIXMsg()’ is made immediately. If unix_errno is not a valid constant, the static buffer will be updated with the appropriate error message.

This tool is primarily intended for users of the C programming language. However, we believe that this functionality will support users of the POSIX FORTRAN language as well. Please refer to POSIX FORTRAN 77 IEEE Std 1003.9-1992 on page 14, Section 2.4 (Error Numbers) for information regarding POSIX FORTRANÕs implementation of standard error return values.

REQUIREMENTS: PGSTK-0582, PGSTK-0600, PGSTK-0632, PGSTK-0650

Set Static Status Message

NAME: **PGS_SMF_SetStaticMsg( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_SetStaticMsg(
        PGSt_SMF_code     code,
        char              *funcname);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_setstaticmsg(code,funcname)
        integer code
        character*32   funcname
```

DESCRIPTION: This tool will provide the means to set a pre-defined error/status message in response to the outcome of some segment of processing.

INPUTS: code-mnemonic error/status code generated by message compiler (see "smfcompile")

funcname-function where the status condition occurred

OUTPUTS: None

RETURNS:

Table 6-38. PGS_SMF_SetStaticMsg Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error message
PGSSMF_E_LOGFILE	Error opening status, report or user files
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
returnStatus =
        PGS_SMF_SetStaticMsg(PGSSMF_E_UNDEFINED_UNIXERROR,
        "My_Function()");
```

FORTRAN:

```
implicit none

integer     returnstatus
integer     pgs_smf_setstaticMsg
```

```
returnstatus =
      pgs_smf_setstaticMsg(PGSSMF_E_UNDEFINED_UNIXERROR,
      ‘my_function()’)
```

NOTES: The parameter “funcname” can be passed in as NULL if you do not wish to record that routine that noted this error. However, it is strongly recommended that you pass the routine name for tracking purposes.

REQUIREMENTS: PGSTK-0582, PGSTK-0600, PGSTK-0650

Set Dynamic Status Message

NAME: **PGS_SMF_SetDynamicMsg( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_SetDynamicMsg(
            PGSt_SMF_code       code,
            char                *msg,
            char                *funcname);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_setdynamicmsg(code,msg,funcname)
            integer         code
            character*240   msg
            character*32    funcname
```

DESCRIPTION: This tool will provide the means to set a runtime specific status message, for a particular status code, in response to the outcome of come segment of processing.

INPUTS: code-mnemonic error/status code generated by message compiler

msg-message string to be saved into the static buffer

funcname-function where the status condition occurred

OUTPUTS: None

RETURNS:

Table 6-39. PGS_SMF_SetDynamicMsg Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error
PGSSMF_E_LOGFILE	Error opening status, report or user files

EXAMPLES:

C: Having defined a mnemonic code in the SMF file:

```
INSTR_E_BAD_CALIBRATION Calibration value %7.2f
            is not within tolerance
```

We would like to insert the calibration factor into the message template during processing, since the value is not fixed prior to runtime. The message that would be set in the status buffer would then appear as:

```
‘Calibration value 356.23 is not within tolerance’

PGSt_SMF_status   returnStatus;
PGSt_SMF_code     code;
char              msg[PGS_SMF_MAX_MSG_SIZE];
char              buf[PGS_SMF_MAX_MSGBUF_SIZE];
float             calibration_factor = 356.23;

calibration_factor = Get_Instrument_Calibration( NIGHT );
/# value of 356.23 returned #/

returnStatus =
PGS_SMF_GetMsgByCode(INSTR_E_BAD_CALIBRATION,msg);
      sprintf(buf,msg,calibration_factor);

PGS_SMF_SetDynamicMsg(INSTR_E_BAD_CALIBRATION,buf,Level1A_Initialization()”)
```

FORTRAN: Having defined a mnemonic code in the SMF file:

```
INSTR_E_BAD_CALIBRATION Calibration value is not
                  within tolerance ->
```

We would like to insert the calibration factor to the end of the message template during processing, since the value is not fixed prior to runtime. The message that would be set in the status buffer would then appear as:

```
‘Calibration value is not within tolerance -> 356.23’

implicit none

integer           pgs_smf_getmsgbycode
integer           pgs_smf_setdynamicmsg
integer           returnstatus
character*240     msg
character*480     buf
real              calibration_factor
integer           msglen
character*8       coeff_str

calibration_factor = get_instrument_calibration( NIGHT )

C   value of 356.23 returned
                    returnstatus = pgs_smf_getmsgbycode(
                      INSTR_E_BAD_CODE,msg)
```

```
write( coeff_str,’(F7.2)’) calibration_factor
msglen = len( msg)
buf = msg(1:msglen)//coeff_str

pgs_smf_setdynamicmsg( INSTR_E_BAD_CALIBRATION, buf,
  ‘level1A_initialization’ );
```

NOTES: Note that you can have the flexibility of associating any dynamic message string to the defined mnemonic code via this routine.

This tool can be used in various situations. For instance the user might want to concatenate some message strings together and assign the resultant string to an existing mnemonic code, so that this message can be passed forward to another module for further processing. Alternatively it can be used to embed runtime variables in the defined message template before saving this message string to the static message buffer.

The parameter “funcname” can be passed in as NULL if you do not wish to record the routine that noted this error. However, it is strongly recommended that you pass the routine name for tracking purposes.

The parameter “msg” can be passed in as NULL. If you do, no message is associated with the mnemonic code.

Refer to utility “smfcompile” for additional information on the format of the message compiler.

REQUIREMENTS: PGSTK-0582, PGSTK-0600, PGSTK-0650

Get Status Message by Code

NAME: **PGS_SMF_GetMsgByCode( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_GetMsgByCode(
        PGSt_SMF_code       code,
        char                msg[]);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_getmsgbycode(code,msg)
        integer         code
        character*240   msg
```

DESCRIPTION: This tool will provide the means to retrieve the message string that is associated with a specific status code in the Status Message Files.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: msg-user pre-defined message string

RETURNS:

Table 6-40. PGS_SMF_GetMsgByCode Returns;

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES: See example for PGS_SMF_SetDynamicMsg( ).

NOTES: This tool provides a simple Status Message File (SMF) lookup function. It should be used primarily for retrieving messages that contain C-style formatting tokens to facilitate the replacement of those tokens with runtime data.

REQUIREMENTS: PGSTK-0580, PGSTK-0650

Get Status Message

NAME: **PGS_SMF_GetMsg( )**

SYNOPSIS

C:

```
#include <PGS_SMF.h>

void
PGS_SMF_GetMsg(
        PGSt_SMF_code       *code,
        char                mnemonic[],
        char                msg[]);
```

FORTRAN:

```
call pgs_smf_getmsg(code,mnemonic,msg)
        integer         code
        character*32        mnemonic
        character*480   msg
```

DESCRIPTION: This tool will provide the means to retrieve status information from the static buffer, for use when reporting on specific status conditions.

INPUTS: None

OUTPUTS: mnemonic-previously set mnemonic error/status string

msg-previously set message string

RETURNS: None

EXAMPLES: See example for PGS_SMF_SetDynamicMsg( ).

NOTES: Until a call is made which sets status information into the buffer, none exists. Therefore, first time calls to this function may return the following for each of the arguments: code=0, mnemonic="", and msg="".

A call to any of the PGS_SMF_Set*( ) functions will load status information into the static buffer. To ensure that the caller of your function can receive the intended information, calls to the PGS_SMF_Set*( ) functions should be performed just prior to returning control back to the caller.

To ensure that the status information received pertains to the status condition set during the last function call, it is imperative that the user invoke this function immediately upon gaining control back from the function that set the status information.

REQUIREMENTS: PGSTK-0580, PGSTK-0650

Get Action Message by Code

NAME: **PGS_SMF_GetActionByCode( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_GetActionByCode(
            PGSt_SMF_code       code,
            char                action[]);
```

FORTRAN:

```
include ‘PGS_SMF.f’

integer function pgs_smf_getactionbycode(code,action)
            integer         code
            character*240   action
```

DESCRIPTION: This tool will provide the means to retrieve an action string corresponding to a specific mnemonic code.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: action-associated action string

RETURNS:

Table 6-41. PGS_SMF_GetActionByCode Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error
PGSSMF_W_NOACTION	No action defined
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char  action[PGS_SMF_MAX_ACT_SIZE];

returnStatus =
PGS_SMF_GetActionByCode(PGSSMF_E_UNDEFINED_UNIXERROR,
      action);
if (returnStatus != PGS_S_SUCCESS)
{
   /# could not retrieve action message #/
}
```

```
else
{
   /# generate a status report and indicate action to be
      taken #/
}
```

FORTRAN:

```
implicit none

integer           pgs_smf_getactionbycode
integer           returnstatus
character*240     action
    returnstatus = pgs_smf_getactionbycode(
      PGSSMF_E_UNDEFINED_UNIXERROR, action );
    IF (returnstatus .NE. PGS_S_SUCCESS) THEN

C   could not retrieve action message
                    ELSE

C   generate status report and indicate action to be taken
                    ENDIF
```

NOTES: This routine will not return any associated action string if the creator of the status code did not associate an action label when creating the Status Message File entry for that status code. If this is the case, the resulting parameter is action[0] = ‘\0’. Refer to the available documentation for the ‘smfcompile’ utility for additional information on how to define and attach action messages to status code entries.

REQUIREMENTS: PGSTK-0591, PGSTK-0650

Create Message Tag

NAME: **PGS_SMF_CreateMsgTag( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_CreateMsgTag(
        char    systemTag[]);
```

FORTRAN:

```
integer function pgs_smf_createmsgtag(systemtag)
        char*60         systemtag
```

DESCRIPTION: The tool described here allows the user to generate a runtime specific character string that may be useful for tagging important items of data. The string contains system defined identifiers that, when combined, can be useful for stamping non-product specific data for system traceability.

INPUTS: None

OUTPUTS: systemTag-system defined message string

RETURNS:

Table 6-42. PGS_SMF_CreateMsgTag Returns

Return	Description
PGS_S_SUCCESS	Success
PGSSMF_W_NO_CONSTRUCT_TAG	No information to construct message tag
PGSSMF_E_BAD_REFERENCE	Bad reference

EXAMPLES:

C:

```
char  systemTag[PGSd_SMF_TAG_LENGTH_MAX];
PGSt_SMF_status returnStatus;

returnStatus = PGS_SMF_CreateMsgTag(systemTag);
if (returnStatus == PGS_S_SUCCESS)
{
   /# create message tag successful #/
}
```

FORTRAN:

```
implicit none

integer     pgs_smf_createmsgtag
char*60     systemtag
integer     returnstatus
```

```
                  returnstatus = pgs_smf_createmsgtag(systemtag)
                      IF (returnstatus .EQ. PGS_S_SUCCESS) THEN

C   create message tag successful
                      ENDIF
```

NOTES: Currently, the only system identifiers used to create the message tag are:

the Science Software Configuration ID,

and the Production Run ID.

IMPORTANT TOOLKIT NOTES

The logical parameter identifiers, which are implicitly defined by the PC tools, are internally mapped to an associated physical parameter through the Process Control mechanism. Therefore before this tool can be used, a Process Control Table MUST be created and properly filled out. In addition, the following environment variables must be set to ensure proper operation:

PGS_PC_INFO_FILEpath to process control file

REQUIREMENTS: PGSTK-0610

Get Instrument Name

NAME: **PGS_SMF_GetInstrName( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_GetInstrName(
        PGSt_SMF_code      code,
        char               instr[]);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_getinstrname(code,instr)
        integer code
        character*10   instr
```

DESCRIPTION: This tool may be used to retrieve the instrument name from a given error/status code.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: instr-corresponding instrument name as it appears in the message text file after the token %INSTR.

RETURNS:

Table 6-43. PGS_SMF_GetInstrName Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES:

C:

```
PGSt_SMF_status          returnStatus;
char  instr[PGS_SMF_MAX_INSTR_SIZE];

returnStatus = PGS_SMF_GetInstrName(MODIS_E_BAD_CALIBRATION
,instr);
if (returnStatus == PGS_S_SUCCESS)
{
   /# record instrument that generated instrument condition
#/
}
```

FORTRAN:

```
implicit none

integer           pgs_smf_getinstrname
integer           returnstatus
character*10instr

returnstatus = pgs_smf_getinstrname(
      MODIS_E_BAD_CALIBRATION, instr )
IF (returnstatus .EQ. PGS_S_SUCCESS) THEN
```

```
C   record instrument which generated status condition
                  ENDIF
```

NOTES: This function may be useful for programs which link in libraries created by cooperating instrument teams, and where the need to distinguish the status conditions associated with each instrument team arises.

REQUIREMENTS: PGSTK-0620, PGSTK-0650

Generate Status Report

NAME: **PGS_SMF_GenerateStatusReport( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_GenerateStatusReport(
        char    *report);
```

FORTRAN:

```
include‘PGS_SMF.f’

integer function pgs_smf_generatestatusreport(report)
        char*1024      report
```

DESCRIPTION: This tool provides the method for the user to create status reports for use by Science Computing Facility personnel. Each call to this procedure causes the user defined report to be appended to the status report log.

INPUTS: report-user report generated text

OUTPUTS: None

RETURNS:

Table 6-44. PGS_SMF_GenerateStatusReport Returns

Return	Description
PGS_S_SUCCESS	Success
PGSSMF_E_LOGFILE	Error opening status, report or user files

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;

returnStatus = PGS_SMF_GenerateStatusReport(“Write it into
status report file”);
if (returnStatus == PGS_S_SUCCESS)
{
   /# write to status report successful #/
}
```

FORTRAN:

```
implicit none

integer     pgs_smf_cgeneratestatusreport
integer     returnStatus
```

```
                    returnStatus = pgs_smf_cgeneratestatusreport(“Write it into
                          status report file”)
                    IF (returnStatus .EQ. PGS_S_SUCCESS) THEN
C   write to status report successful
                    ENDIF
```

NOTES: The system defined message tag will automatically be added to the user-provided report.

IMPORTANT TOOLKIT NOTES

The logical file identifier (PGSd_SMF_LOGICAL_LOGSTATUS), which is implicitly used by this tool, is internally mapped to an associated physical file through the Process Control mechanism. Therefore before this tool can be used, a Process Control Table MUST be created and properly filled out. In addition, the following environment variables must be set to ensure proper operation:

Table 6-45. Environment Variables

Variable	Path
PGS_PC_INFO_FILE	path to process control file

REQUIREMENTS: PGSTK-0650

Send Runtime Data

NAME: **PGS_SMF_SendRuntimeData( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_SendRuntimeData(
        PGSt_integer  numfiles,
        PGSt_integer  files[])
        PGSt_integer  version[];
```

FORTRAN:

```
include ‘PGS_SMF.f’

integer function pgs_smf_sendruntimedata(numfiles,files,version)
        integer numfiles
        integer files(*)
        integer version(*)
```

DESCRIPTION: This tool provides the user with a method for flagging specific runtime data files for subsequent post-processing retrieval.

INPUTS: numfiles-exact number of runtime logical file identifiers loaded into the array ‘files’

files-array of logical file identifiers which are to be preserved for later retrieval

version-an associated array for identifying specific versions of the files identified in the preceding array of logical identifiers

OUTPUTS: None

RETURNS:

Table 6-46. PGS_SMF_SendRuntimeData Returns

Return	Description
PGS_S_SUCCESS	Success
PGSSMF_E_SENDRUNTIME_DATA	Send runtime file data error
PGSSMF_M_TRANSMIT_DISABLE	Transmission of files is disabled

EXAMPLES:

C:

```
==
/# These constants may be defined in the users include
   file(s). #/
```

```
/# Note that these logical file identifiers would have to
   appear #/
/# in the Process Control file in order for this call to
   work. #/
#define MODIS1A   10
#define MODIS2    20
#define TEMP1     50
#define TEMP2     51
#define TEMP3     52

PGSt_SMF_status   returnStatus;
PGSt_integernumberOfFiles;
PGSt_integerlogIdArray[6];
PGSt_integerversion[6];
PGSt_integerversion_MODIS1A_1 = 1;
PGSt_integerversion_MODIS1A_2 = 2;
PGSt_integerversion_MODIS2    = 1;
PGSt_integer      version_TEMP      = 1;

logIdArray[0] = MODIS1A;  version[0] = version_MODIS1A_1;
logIdArray[1] = MODIS1A;  version[1] = version_MODIS1A_2;
logIdArray[2] = MODIS2;   version[2] = version_MODIS2;
logIdArray[3] = TEMP1;    version[3] = version_TEMP;
logIdArray[4] = TEMP2;    version[4] = version_TEMP;
logIdArray[5] = TEMP3;    version[5] = version_TEMP;
numberOfFiles = 6;

returnStatus =
PGS_SMF_SendRuntimeData(numberOfFiles,logIdArray,version);
if (returnStatus == PGS_S_SUCCESS)
{
    /# send runtime data success #/
}
```

FORTRAN:

```
C         The following constants may be defined in the users include file(s).

C         Note that the specific logical file identifiers would have to appear

C         in the process control file in order for this call to work.

implicit none

                  integer     pgs_smf_sendruntimedata
                  integer     modis1a
                  parameter   (modis1a = 10)
                  integer     modis2
                  parameter   (modis2 = 20)
                  integer     temp1
                  parameter   (temp1 = 50)
                  integer     temp2
                  parameter   (temp2 = 51)
                  integer     temp3
                  parameter   (temp2 = 52)

                  integer     returnStatus
                  integer     numberOfFiles
                  integer     logIdArray(6)
                  integer     version(6)
                  integer     version_modis1a_1
                  integer     version_modis1a_2
                  integer     version_modis2
                  integer     version_temp

                  version_modisa_1  = 1
                  version_modisa_2  = 2
                  version_modis2    = 1
                  version_temp= 1

                  logIdArray(1)     = modis1a
                  version(1)        = version_modis1a_1

                  logIdArray(2)     = modis1a
                  version(2)        = version_modis1a_2

                  logIdArray(3)     = modis2
                  version(3)        = version_modis2

                  logIdArray(4)     = temp1
                  version(4)        = version_temp

                  logIdArray(5)     = temp2
                  version(5)        = version_temp

                  logIdArray(6)     = temp3
                  version(6)        = version_temp

                  numberOfFiles     = 6

return_status = pgs_smf_sendruntimedata(numberOfFiles,logIdArray,version)

if (return_status .EQ. PGS_S_SUCCESS) then

C         send runtime data success
                  endif
```

NOTES: Repeated calls to this tool will cause previously requested files to be superseded with the list provided during the last call.

IMPORTANT TOOLKIT NOTES

This tool does not trigger the spontaneous transmission of runtime files and e-mail notification, as it did in Toolkit 3. Rather, the requested files are saved/marked for transmission following the normal termination of the PGE process. The actual transmission procedure is performed by the termination process (See PGS_PC_TermCom() for more information on the steps required to perform this transmission).

Please refer to the documentation for PGS_PC_TermCom() for directions on how to activate/deactivate the Toolkit's transmission capability.

REQUIREMENTS: PGSTK-0630

Test Error Level

NAME: **PGS_SMF_TestErrorLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestErrorLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testerrorlevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'E'.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_SMF_boolean  levelFlag;
int               *intPtr;

returnStatus = PGS_MEM_Malloc(&intPtr,sizeof(int)*10);
levelFlag = PGS_SMF_TestErrorLevel(returnStatus);
if (levelFlag
if (PGS_SMF_TestErrorLevel(returnStatus) == PGS_TRUE)
{
   /# Branch to handle error condition #/
}
else
{
   /# Some other status level returned #/
}
```

FORTRAN:

```
                implicit none

                INTEGER     pgs_pc_getnumberoffiles
                INTEGER     returnstatus
                INTEGER     numfiles
                INTEGER     levelflag
                PARAMETER   (ceres4 = 7090)
                INTEGER     ceres4

                returnstatus = pgs_pc_getnumberoffiles(ceres4,numfiles)
                levelflag = pgs_smf_testerrorlevel(returnstatus)
                IF (levelflag .EQ. PGS_TRUE) THEN
C      Branch to handle error condition
                ELSE
C      Some other status level returned
                ENDIF
```

NOTES: None

REQUIREMENTS: PGSTK-0590

Test Fatal Level

NAME: **PGS_SMF_TestFatalLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestFatalLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include‘PGS_SMF.f’

integer function pgs_smf_testfatallevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level ‘F’.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

NOTES: NONE

EXAMPLES: See example for PGS_SMF_TestErrorLevel( );

NOTES: None

REQUIREMENTS: PGSTK-0590

Test Message Level

NAME:	**PGS_SMF_TestMessageLevel( )**
SYNOPSIS:	

C:	#include <PGS_SMF.h>
	
	PGSt_SMF_boolean
	PGS_SMF_TestMessageLevel(
	          PGSt_SMF_status code);

FORTRAN:	include‘PGS_SMF.f’ integer function pgs_smf_testMessagelevel(code) integer code
DESCRIPTION:	Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level ‘M’.
INPUTS:	code-mnemonic error/status code generated by message compiler
OUTPUTS:	None
RETURNS:	PGS_FALSE PGS_TRUE
NOTES:	None
EXAMPLES:	See example for PGS_SMF_TestErrorLevel( );
REQUIREMENTS:	PGSTK-0590

Test Warning Level

NAME: **PGS_SMF_TestWarningLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestWarningLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testwarninglevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level ‘W’.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

NOTES: None

EXAMPLES: See example for PGS_SMF_TestErrorLevel( );

REQUIREMENTS: PGSTK-0590

Test User Information Level

NAME:	**PGS_SMF_TestUserInfoLevel( )**
SYNOPSIS:	
C:	#include <PGS_SMF.h>

```
PGSt_SMF_boolean
PGS_SMF_TestUserInfoLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testuserinfolevel(code)
        integer code
```

DESCRIPTION:	Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'U'.
INPUTS:	code-mnemonic error/status code generated by message compiler
OUTPUTS:	None
RETURNS:	PGS_FALSE PGS_TRUE
EXAMPLES:	See example for PGS_SMF_TestErrorLevel( );
NOTES:	None
REQUIREMENTS:	PGSTK-0590

Test Success Level

NAME:	**PGS_SMF_TestSuccessLevel( )**
SYNOPSIS:	
C:	#include <PGS_SMF.h>
	PGSt_SMF_boolean PGS_SMF_TestSuccessLevel(PGSt_SMF_status code);
FORTRAN:	include'PGS_SMF.f'
	integer function pgs_smf_testsuccesslevel(code) integer code
DESCRIPTION:	Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'S'.
INPUTS:	code-mnemonic error/status code generated by message compiler
OUTPUTS:	None
RETURNS:	PGS_FALSE PGS_TRUE
EXAMPLES:	See example for PGS_SMF_TestErrorLevel( );
NOTES:	None
REQUIREMENTS:	PGSTK-0590

Test Notice Level

NAME:	**PGS_SMF_TestNoticeLevel( )**
SYNOPSIS:	
C:	#include <PGS_SMF.h>
	PGSt_SMF_boolean PGS_SMF_TestNoticeLevel(PGSt_SMF_status code);
FORTRAN:	include'PGS_SMF.f'
	integer function pgs_smf_testnoticelevel(code) integer code
DESCRIPTION:	Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'N'.
INPUTS:	code-mnemonic error/status code generated by message compiler
OUTPUTS:	None
RETURNS:	PGS_FALSE PGS_TRUE
EXAMPLES:	See example for PGS_SMF_TestErrorLevel( );
NOTES:	None
REQUIREMENTS:	PGSTK-0590

Test Status Level

NAME: **PGS_SMF_TestStatusLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_TestStatusLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_teststatuslevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a defined status level constant.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS:

Table 6-47. PGS_SMF_TestStatusLevel Returns

Return	Description
PGS_SMF_MASK_LEV_S	Success level status
PGS_SMF_MASK_LEV_M	Message level status
PGS_SMF_MASK_LEV_U	User information level status
PGS_SMF_MASK_LEV_N	Notice level status
PGS_SMF_MASK_LEV_W	Warning level status
PGS_SMF_MASK_LEV_E	Error level status
PGS_SMF_MASK_LEV_F	Fatal level status
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
int               *intPtr;

returnStatus = PGS_MEM_Malloc(&intPtr,sizeof(int)*10);
switch(PGS_SMF_TestStatusLevel(returnStatus))
{
   case PGS_SMF_MASK_LEV_S:
   /# This is a success level status #/
      break;

   case PGS_SMF_MASK_LEV_M:
   /# This is a message level status #/
      break;

   case PGS_SMF_MASK_LEV_U:
   /# This is a user information level status #/
      break;

   case PGS_SMF_MASK_LEV_N:
   /# This is a notice level status #/
      break;

   case PGS_SMF_MASK_LEV_W:
   /# This is a warning level status #/
      break;

   case PGS_SMF_MASK_LEV_E:
   /# This is a error level status #/
      break;

   case PGS_SMF_MASK_LEV_F:
   /# This is a fatal level status #/
      break;

   default:
   /# Undefined status level #/
      break;
}
```

FORTRAN:

```
                  implicit none

                  INTEGER            pgs_pc_getnumberoffiles
                  INTEGER            returnstatus
                  INTEGER            numfiles
                  INTEGER            levelmask
                  PARAMETER          (ceres4 = 7090)
                  INTEGER            ceres4

                  returnstatus = pgs_pc_getnumberoffiles(ceres4,numfiles)
                  levelmask = pgs_smf_teststatuslevel(returnstatus)
                  IF (levelmask .EQ. PGS_SMF_MASK_LEV_S) THEN
C   This is a success level status
                  ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_M) THEN
C   This is a message level status
                  ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_U) THEN
```

```
C   This is a user information level status
                  ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_N) THEN

C   This is a notice level status
                  ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_W) THEN

C   This is a warning level status
                  ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_E) THEN

C   This is a error level status
                  ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_F) THEN

C   This is a fatal level status
                  ELSE

C   Undefined status level
                  ENDIF
```

NOTES: The returned level constants are ordered by severity with PGS_SMF_MASK_LEV_S having a small integral value and PGS_SMF_MASK_LEV_F having the highest. This enables you to perform conditional tests between a particular status code and one of the provided level constants.

REQUIREMENTS: PGSTK-0590

Begin Function

NAME: **PGS_SMF_Begin( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>


PGSt_SMF_status
PGS_SMF_Begin(
        char *funcname);
```

FORTRAN:

```
include ‘PGS_SMF.f’

integer function pgs_smf_begin(funcname)
character*100 funcname
```

DESCRIPTION: A call to this tool signals to SMF that a function has started, and thus, the current message indent level should be incremented.

INPUTS:

Table 6-48. PGS_SMF_Begin Returns

Name	Description
funcname	The name of the function which calls this routine.

OUTPUTS: **NONE**

RETURNS: PGS_S_SUCCESS

EXAMPLES:

C:

```
PGSt_SMF_status returnStatus;

returnStatus = PGS_SMF_Begin(“CallingFunction”);
```

FORTRAN:

```
integer pgs_smf_begin

integer returnStatus

returnStatus = pgs_smf_begin(‘CallingFunction’)
```

NOTES: A message will be written to the status log file indicating that the specified function has started.

REQUIREMENTS: PGSTK-0580,0590,0650,0663

End Function

NAME: **PGS_SMF_End( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_End(
        char *funcname);
```

FORTRAN:

```
include ‘PGS_SMF.f’

integer function pgs_smf_end(funcname)
character*100 funcname
```

DESCRIPTION: A call to this tool signals to SMF that a function has completed, and thus, the current message indent level should be decremented.

INPUTS:

Table 6-49. PGS_SMF_End Returns

Name	Description
funcname	The name of the function which calls this routine.

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS

EXAMPLES:

C:

```
PGSt_SMF_status returnStatus;

returnStatus = PGS_SMF_End(“CallingFunction”);
```

FORTRAN:

```
implicit none

integer pgs_smf_end

integer returnStatus

returnStatus = pgs_smf_end(‘CallingFunction’)
```

NOTES: A message will be written to the status log file indicating that the specified function has completed.

REQUIREMENTS: PGSTK-0580,0590,0650,0663

Set Arithmetic Trap

We have found that this function could not be implemented in a POSIX compliant manner across all development platforms. We note, however, that with the exception of one platform (IBM), all machines, by default, enable their own implementation-dependent floating-point exception handling features. In a general sense, these features provide the functional equivalent of the Toolkit exception handling mechanism. See “Investigation Results on the use of Signal Exception Handling for ECS Approved Computing Platforms” on the Toolkit Primer web page for more details.

NAME: **PGS_SMF_SetArithmeticTrap( )**

SYNOPSIS:

C:

```
#include <PGSSMF.h>

PGSt_SMF_status
PGS_SMF_SetArithmeticTrap(
        void (*func)(int signo));
```

FORTRAN: TBD

DESCRIPTION: This tool should be used to specify a signal handling function to be called to handle arithmetic exception events.

INPUTS: func-signal handling function

OUTPUTS: None

RETURNS:

Table 6-50. PGS_SMF_SetArithmeticTrap Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error

EXAMPLES:

C:

```
PGSt_SMF_status returnStatus;

void SignalHandler(int signo)
{
   /# algorithm to handle SIGFPE #/
}
```

```
main( )
{
   /# initialization section #/

   returnStatus = PGS_SMF_SetArithmeticTrap(SignalHandler);
   if (returnStatus == PGS_S_SUCCESS)
   {
      /# signal trap set successfully #/
   }
   else
   {
      /# signal trap not set #/
      exitStatus = 1;
      goto EXIT;
   }
   /# main body #/
   .
   .
   .
   for (alt=5000; alt<100000; alt+500)
   {
      density[alt]=(GAS_CONST * temp[alt]) / pressure[alt];
   }
   .
   .
   .
   EXIT:
   exit( existStatus );
}     /# end main #/
```

FORTRAN: TBD

NOTES: Use NULL in place of a signal handling function to set the Toolkit default signal handling function. This handler will force an exit from the user's program, which is generally more acceptable than the system's default action (i.e., core dump).

Upon successful completion of the user's signal handling function, program control will be returned to the point where the fault occurred. As a side-effect, the default Toolkit signal handling function will be restored to safeguard against future occurrences of this event.

The user's signal handling routine must accept the integer argument for the signal number. It is not required for the user to take any action on the value; it is strictly for informational purposes only.

This tool only responds to the POSIX signal SIGFPE; all other signals need to be handled by other means.

REQUIREMENTS: PGSTK-0660

6.2.2.3 Error and Status Message File Creation Tool

Status Message File Creation

NAME:	**smfcompile**
SYNOPSIS:	
C:	smfcompile -f textfile [-r] [-i]
FORTRAN:	smfcompile -f textfile -f77 [-r] [-i]
ALL:	smfcompile -f textfile -all [-r] [-i]
Ada:	smfcompile -f textfile -ada [-r] [-i]
DESCRIPTION:	This utility generates runtime status message files and language dependent include files from user-defined status message text files.
INPUTS:	textfile-status message text file (e.g., PGS_IO_100.t)
•	f77-create FORTRAN include file
•	all-create FORTRAN, C and Ada include files
•	r-redirect the created ASCII runtime message file to the directory set in the environment variable “PGSMSG”
•	i-redirect the created language-specific include file to the directory set in the environment variable “PGSINC”
OUTPUTS:	Language-specific include file and ASCII runtime message file (an Ada package specification will be produced in place of an include file when the ‘-ada’ switch is used).
RETURNS:	1-error occurred
	0-successful operation
EXAMPLES:	smfcompile -f PGS_IO_100.t (produces PGS_IO_100.h and PGS_100)
	smfcompile -f PGS_IO_100.t -f77 (produces PGS_IO_100.f and PGS_100)
	smfcompile -f PGS_IO_100.t -all (produces PGS_IO_100.f, PGS_IO_100.h, PGS_IO_100.a and PGS_100)
NOTES:	The environment variable PGSMSG must be set to the local Toolkit installation directory ‘/../pgs/message’ in order for the Toolkit to function properly. The reason for this is that Toolkit status message files will

already reside in this directory upon completion of the Toolkit installation procedure; these files must be visible at runtime for the Toolkit to function properly.

If you do not specify the "-r" input parameter to the smfcompile, then make sure that the newly created ASCII runtime message file is moved to the directory set in the environment variable "PGSMSG".

REQUIREMENTS: PGSTK-0581, PGSTK-0590, PGSTK-0591, PGSTK-0600, PGSTK-0650, PGSTK-0664

6.2.3 Process Control Tools

The Process Control Tools perform the task of communicating Process Control information to the PGE. This information may consist of Production Run ID; Science Software ID; physical file names (or *Universal Reference* identifiers); input file metadata/ attributes; and PGE specific runtime parameter information. Access to this data is provided through a library API and a command-level interface, as described in detail below.

For Toolkit 5, an additional tool has been created which allows the user to query on the type of file that is of current interest. This tool, PGS_PC_GetReference, provides the user with the means to determine whether a file is of type temporary or product.

Another important change for Toolkit 5 involves the removal of most Toolkit dependency information based on environment variables. All the environment variables that define the default location for PCF information, for each PCF section (e.g., product input), have been replaced with section headers in the PCF. The means to provide this default information is still there, but the method has been changed. To reduce the number of environment variables that the user would otherwise, as in the past, be required to set.

Several new tools were added for Toolkit 4; chief among them was the product metadata retrieval tools PGS_PC_GetFileAttr and PGS_PC_GetFileByAttr. These tools provide the means to retrieve metadata that results from an inventory search; a search performed, by the Planning and Data Processing subsystem, as part of the normal processing setup prior to PGE execution. These tools should not be confused with the Metadata tools that are more specialized tools for managing the various types of metadata (See Section 6.2.1.4). These latter tools provide for the generation and association of product metadata whereas the former only provide for the retrieval of product metadata. Once the definition for metadata matures and the design for managing it in the data server becomes clearer, it may be possible to unify these tools in such a way as to provide for the greatest degree of benefit to the user.

In addition to the above, several new tools were added in Toolkit 4 to provide command, or shell, level access to most of the process control functionality delivered in Toolkit 3. This additional interface will provide for a greater degree of flexibility, when developing PGEs, by allowing the user to take advantage of standard shell level features when manipulating process control information.

However, some of these new tools have a different objective. To provide for a more seamless integration of the Toolkit with a PGE, a few command utilities have been incorporated which perform Toolkit initialization and termination procedures; these steps are necessary to support the Toolkit to its fullest extent. Since these tools are used outside of the PGE, they do not place an additional burden on the development of a PGE. The user is however encouraged to activate these tools whenever testing is performed. To provide for this eventuality, there is now a shell command that provides an integrated solution for the inclusion of these tools during PGE testing.

As newer, higher-level, tools have emerged, greater has the need become to abstract away the older, lower-level tools. To safeguard against future changes in the Toolkit API, the PGS_PC_GetPCSData and PGS_PC_PutPCSData routines were removed from the User's Guide in Toolkit 4. This step is necessitated by the possibility of having to support a different Process Control implementation for the DAAC environment. We regret any inconvenience that this may cause.

In order for these tools to function, the actual process control information needs to be specified in a Process Control file (PCF) prior to activation of the PGE. Each Process Control file contains various subject fields to hold specific runtime information. All product/support/temporary file I/O subject fields follow a similar format; the ones that differ deal with system defined and user defined parameter information. Each subject-field entry contains a key identifier and numerous attributes that describe the particular entry.

To support testing of a PGE, the user must create entries in a PCF to account for all file inputs, all file outputs (except intermediate and temporary), and all parameter information that the particular PGE depends on. The key identifiers that name each entry, also need to be represented as logical identifiers in the PGE software. Then at runtime, the attributes for a particular entry may be retrieved by passing a specific key identifier to the appropriate PC Toolkit function. (Note that certain IO Toolkit functions access the file I/O entries when product/support/temporary file key identifiers are passed to them) For this reason, it would be prudent to create a meaningful constant identifier for each key identifier in the PCF, e.g., TEMP1=100.

This process of defining a PCF will need to be performed for every unique instance of a PGE. At runtime, these tools will access the particular PCF that is pointed to by the environment variable PGS_PC_INFO_FILE.

The measures outlined in the preceding paragraph must be performed to provide the minimal level of PGS emulation required to support the Toolkit, since many Toolkit functions rely on the Process Control mechanism for I/O and parameter information. The Process Control File ‘PCF.v5,’ which was delivered along with the Toolkit in directory ‘$PGSHOME/runtime,’ contains all the necessary Toolkit dependencies, some of which may need to be customized for certain Toolkit functions. **To avoid PCF collisions between Toolkit and developer dependencies, logical identifiers in the range 10,000 to 10,999 have been reserved exclusively for Toolkit use; any other valid positive integer may be used for development purposes.**

To mediate against any potential problems caused by an improperly constructed Process Control File; an additional tool has been added which can be used by the developer to screen a PCF for syntax errors and missing Toolkit dependencies. For more information on the usage of this utility, refer to the section below for the ‘pccheck’ tool.

Please refer to Appendix C for guidance on the construction of Process Control Files and to examine a sample PCF. More details and examples on the usage of the ‘pccheck’ utility are also included in this appendix.

6.2.3.1 Process Control Command Tools

Toolkit Shell Script Command

NAME:	**PGS_PC_Shell.sh**
SYNOPSIS:	PGS_PC_Shell.sh [-h] <PGE file> <Init string> <PCF location> <SMF Cache Size> [-v] [-p]
C:	N/A
FORTRAN:	N/A
DESCRIPTION:	This shell script accepts four command line arguments as input. The first argument is the PGE to run. This may be a shell script or an executable. The second argument is the Init string that contains 4 binary digits that define how the Toolkit will behave. Together, these instruct the shell about what to do in the case of using/not using shared memory or using/not using log files. The third argument is the location of the Process Control File (PCF). The forth argument is the SMF cache size. A fifth argument may be used to run this script in verbose mode. A sixth argument may be used to pass the return value of the PGE through as the return value of the script.
INPUTS:	PGE file-The full path/file name of the PGE to be run
	Init string-The string to be passed in with the instructions about what to do with shared memory and the log file. See NOTES section for complete description of each field in the Init string flag.

PCF location-The full path/file name of the Process Control File (PCF)

SMF Cache Size-size of SMF message cache in records

- v-Run in verbose mode. Output status messages displaying settings, current file being run.
- p-Make the return value of this script be the return value of the PGE if the PGE is run. If the PGE does not get run then revert to the normal method of return values for this shell.
- h-Upon receiving the -h flag a short description of the usage of PGS_PC_Shell.sh will be provided to the user and the command will exit.

OUTPUTS:	NONE
RETURNS:	PGS_S_SUCCESS PGS_SH_SYS_PARAM

PGS_SH_MEM_INIT
PGS_SH_PC_DELETETMP
PGS_SH_SMF_SENDRUNTIME
PGS_SH_SMF_SENDLOGFILE
PGS_SH_MEM_TERM
PGS_SH_SMF_LOGFILE
PGS_SH_PC_LOADDATA
PGS_SH_PC_ENV
PGS_SH_SMF_SHMMEM

EXAMPLES:

```
PGS_PC_Shell.sh -h
PGS_PC_Shell.sh /usr/PGE/somePGE 1111
        /usr/PGE/data/PCF.current 50 -v
PGS_PC_Shell.sh /usr/home/PGE/runFile 1010
        /home/PCFDATA/pcf.data 200
PGS_PC_Shell.sh /usr/PGEhome/runThis 0000
        /home/Data/MY.pcf 150 -p
```

NOTES: This shell script parses the input to ensure correctness and will report any input problems to the user.

This shell script acts as the outer most shell for the PGE.

The Init string flag consists of four (4) fields. Each field contains a single digit. The digits should be a one (1) or a zero (0). Therefore the Init String would appear as “1010” or “1111”, etc. For ease of use PGS_PC_Shell.sh will interpret any non-zero digit as a one. Therefore, 8020 would be interpreted as 1010, and 5500 would be interpreted as 1100, etc. The field descriptions are listed as follows:

FIELD 1 - 1 (or any non-zero digit) = Use shared memory if
1 available
0 = Do not use shared memory

FIELD 2 - 1 (or any non-zero digit) = If shared memory fails continue using ASCII files
0 = If shared memory fails stop now

FIELD 3 - 1 (or any non-zero digit) = Use Log Files
0 = Do not use Log Files

FIELD 4 - 1 (or any non-zero digit) = If Log Files fail continue anyway
0 = If Log Files fail stop now

REQUIREMENTS: PGSTK-1312

Toolkit Initialization Command

NAME:	**PGS_PC_InitCom**
SYNOPSIS:	PGS_PC_InitCom <shared-memory-flag> <log-file-flag> <num.-smf-records>
C:	N/A
FORTRAN:	N/A
DESCRIPTION:	This program performs the initialization for the PGE.
INPUTS:	argc-number of command line arguments
	argv[0]-executable name (not processed but listed here anyway)
	argv[1]-flag stating whether or not to use shared memory
	argv[2]-flag stating whether or not to write to a log file
	argv[3]-number of SMF records to store in shared memory
OUTPUTS:	NONE
RETURNS:	PGS_S_SUCCESS PGS_SH_MEM_INIT PGS_SH_SMF_LOGFILE PGS_SH_PC_LOADDATA PGS_SH_PC_ENV PGS_SH_SMF_SHMMEM

EXAMPLES:	`PGS_PC_InitCom ShmOn LogOn 50`
	`PGS_PC_InitCom ShmOff LogOn 100`

NOTES:	This program is intended to be run from within PGS_PC_Shell.sh and is not designed to be run from the command line as a stand-alone program.
REQUIREMENTS:	PGSTK-1311

Get Physical File Reference Command

NAME: **PGS_PC_GetReferenceCom**

SYNOPSIS: PGS_PC_GetReferenceCom <logical ID> <version>

DESCRIPTION: This program will retrieve the physical file reference associated with a logical ID.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-logical ID of the configuration parameter

argv[2]-version of the physical file reference to retrieve. A one-to-one relationship exists between all files except for product input files.

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_NODATA
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297
Version=1

Get the physical file reference associated
#  with ID 12297

REFERENCE=`PGS_PC_GetReferenceCom $LogicalID $Version`
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  This is how the file name and versions remaining
#  can be parsed.
      FILENAME=`echo $REFERENCE | cut -f1 -d” “`
      VERSIONS=`echo $REFERENCE | cut -f2 -d” “`
#  FILENAME now contains the file reference.
#  VERSIONS now contains the versions remaining.
else
```

```
#  report an error found
fi
.
.
.
```

Another method of performing this task is as listed below. This method only works in the Korn and Bourne shells.

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297
Version=1

#  Get the physical file reference associated
#  with ID 12297
set `PGS_PC_GetReferenceCom $LogicalID $Version`
#  The file reference and versions remaining will
#  now appear in two separate tokens.
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
      FILENAME=$1
      VERSIONS=$2
#  FILENAME now contains the file reference.
#  VERSIONS now contains the versions remaining.
else
#  report an error found
fi
.
.
.
```

A final method of performing this task is as listed below. This method only works in the Korn and Bourne shells.

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297
Version=1

#  Get the physical file reference associated
#  with ID 12297
set “`PGS_PC_GetReferenceCom $LogicalID $Version`”
```

```
#  Placing double quotes around the command causes
#  the string to be placed in one token.
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  This is how the file name and versions remaining
#  can be parsed.
      FILENAME=`echo $1 | cut -f1 -d” “`
      VERSIONS=`echo $1 | cut -f2 -d” “`
#  FILENAME now contains the file reference.
#  VERSIONS now contains the versions remaining.
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE script.

The user will be required to parse the file name and number of files remaining from the output string. This can be done using the cut command (See EXAMPLES). The file name and versions remaining will be separated by a single space.

REQUIREMENTS: PGSTK-1290

Get User Defined Configuration Parameters Command

NAME:	**PGS_PC_GetConfigDataCom**
SYNOPSIS:	PGS_PC_GetConfigDataCom <logical ID>
DESCRIPTION:	This program will retrieve user defined configuration parameters from the PCF or shared memory at the command line.
INPUTS:	argc-number of command line arguments argv[0]-executable name (not processed but listed here anyway) argv[1]-logical ID of the configuration parameter
OUTPUTS:	NONE
RETURNS:	PGS_S_SUCCESS PGS_SH_SYS_PARAM PGS_SH_PC_NODATA PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297

#  Get the parameter associated with ID 12297
CONFIG=`PGS_PC_GetConfigDataCom $LogicalID`
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

REQUIREMENTS: PGSTK-1291

Get Number Of Files Command

NAME:	**PGS_PC_GetNumberOfFilesCom**
SYNOPSIS:	PGS_PC_GetNumberOfFilesCom <logical ID>
DESCRIPTION:	This program will retrieve the number of product input files from the PCF or shared memory at the command line.
INPUTS:	argc-number of command line arguments argv[0]-executable name (not processed but listed here anyway) argv[1]-logical ID of the product input files to be inquired
OUTPUTS:	NONE
RETURNS:	PGS_S_SUCCESS PGS_SH_SYS_PARAM PGS_SH_PC_NODATA PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297

#  Get the number of product files associated
#  with ID 12297
NUMFILES=`PGS_PC_GetNumberOfFilesCom $LogicalID`
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

REQUIREMENTS: PGSTK-1315

Get File Attribute Command

NAME:	**PGS_PC_GetFileAttrCom**
SYNOPSIS:	PGS_PC_GetFileAttrCom <logical ID> <version> <format flag>
DESCRIPTION:	This program will retrieve a file attribute string or location associated with a product input file from the PCF or shared memory at the command line.
INPUTS:	argc-number of command line arguments argv[0]-executable name (not processed but listed here anyway) argv[1]-logical ID of the configuration parameter argv[2]-version number of file to retrieve attribute for argv[3]-format flag that states whether to return the attribute or the location of the file attribute. Possible values are: PGSd_PC_ATTRIBUTE_LOCATION PGSd_PC_ATTRIBUTE_STRING
OUTPUTS:	NONE
RETURNS:	PGS_S_SUCCESS PGS_SH_SYS_PARAM PGS_SH_PC_NODATA PGS_SH_PC_TOOLERROR PGS_SH_PC_TRUNC
EXAMPLES:	The following example is valid for the Bourne and Korn shells only.

```
#  This is within a shell script - probably within the
#  PGE script.
#  Set our format flag values. (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
: ${PGSd_PC_ATTRIBUTE_LOCATION=1}
: ${PGSd_PC_ATTRIBUTE_STRING=2}

LogicalID=12297
Version=1
FormatFlag=$PGSd_PC_ATTRIBUTE_STRING

#  Get the file attribute string associated with
#  the first file of product ID 12297
ATTR=`PGS_PC_GetFileAttrCom $LogicalID $Version $FormatFlag`
RETVAL=$?
```

```
#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  Variable ATTR now contains the attribute string
else
#  report an error found
fi
.
.
.
```

If the user wishes to use a c-shell script this is the recommended technique to use. In a c-shell script if the user fails to use this technique the script will give undefined results (see NOTES).

```
#  This is within a shell script - probably within the
#  PGE script.
#  Set our format flag values. (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
set PGSd_PC_ATTRIBUTE_LOCATION=1
set PGSd_PC_ATTRIBUTE_STRING=2

set LogicalID=12297
set Version=1
set FormatFlag=$PGSd_PC_ATTRIBUTE_STRING

#  Get the file attribute string associated with
#  the first file of product ID 12297
PGS_PC_GetFileAttrCom $LogicalID $Version $FormatFlag
>out.file
set RETVAL=$status

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  File out.file now contains the attribute string
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

If the format flag passed in is equal to PGSd_PC_ATTRIBUTE_STRING the return value is the attribute string appended as one long string. If the format flag passed in is equal to PGSd_PC_ATTRIBUTE_LOCATION the return value is the attribute location that is a full path and file name of the file containing the attribute string.

If the user wishes to use this program in a c-shell script the output of the program must be re-directed to a file and the file can then be manipulated. A long string can not be assigned to a variable in a c-shell script. Attempting to assign a long string to a variable will give undefined results in the c-shell.

REQUIREMENTS: PGSTK-1314

Get the Temporary File Reference Command

NAME: **PGS_PC_GetTempReferenceCom**

SYNOPSIS: PGS_PC_GetTempReferenceCom <logical ID> <duration of file>

DESCRIPTION: This program will retrieve a temporary file reference from the PCF. If a reference does not exist it will create one.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-logical ID of the temporary file reference

argv[2]-file duration

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

#  Set our endurance values.  (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
: ${PGSd_IO_Gen_NoEndurance=0}
: ${PGSd_IO_Gen_Endurance=1}

LogicalID=12297
Endurance=$PGSd_IO_Gen_NoEndurance

#  Get the temporary physical file reference associated
#  with ID 12297
TEMPREFERENCE=`PGS_PC_GetTempReferenceCom $LogicalID
$Endurance`
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  This is how the file name and existence flag
#  can be parsed.
      FILENAME=`echo $TEMPREFERENCE | cut -f1 -d” “`
      EXISTS=`echo $TEMPREFERENCE | cut -f2 -d” “`
```

```
#  FILENAME now contains the file reference.
#  EXISTS now contains the existence flag.
else
#  report an error found
fi
.
.
.
```

Another method of performing this task is as listed below. This method only works in the Korn and Bourne shells.

```
#  This is within a shell script - probably within the
#  PGE script.

#  Set our endurance values.  (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
: ${PGSd_IO_Gen_NoEndurance=0}
: ${PGSd_IO_Gen_Endurance=1}

LogicalID=12297
Endurance=$PGSd_IO_Gen_NoEndurance

#  Get the temporary physical file reference associated
#  with ID 12297
set `PGS_PC_GetTempReferenceCom $LogicalID $Endurance`
#  The file reference and existence flag will
#  now appear in two separate tokens.
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
        FILENAME=$1
        EXISTS=$2
#  FILENAME now contains the file reference.
#  EXISTS now contains the existence flag.
else
#  report an error found
fi
.
.
.
```

A final method of performing this task is as listed below. This method only works in the Korn and Bourne shells.

```
#  This is within a shell script - probably within the
#  PGE script.

#  Set our endurance values.  (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
: ${PGSd_IO_Gen_NoEndurance=0}
: ${PGSd_IO_Gen_Endurance=1}

LogicalID=12297
Endurance=$PGSd_IO_Gen_NoEndurance

#  Get the temporary physical file reference associated
#  with ID 12297
set “`PGS_PC_GetTempReferenceCom $LogicalID $Endurance`”
#  Placing double quotes around the command causes
#  the string to be placed in one token.
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  This is how the file name and versions remaining
#  can be parsed.
        FILENAME=`echo $1 | cut -f1 -d” “`
        EXISTS=`echo $1 | cut -f2 -d” “`
#  FILENAME now contains the file reference.
#  EXISTS now contains the existence flag.
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

If a temporary file reference does not exist for the logical ID then a reference is created. The user will be able to determine if the reference existed by checking the existence flag portion of the program return (See EXAMPLES).

The user will be required to parse the file name and the existence flag from the output string. This can be done using the cut command (See EXAMPLES). The file name and the existence flag will be separated by a single space.

REQUIREMENTS: PGSTK-0531, PGSTK-0535, PGSTK-1291

Delete Temporary File Command

NAME:	**PGS_PC_TempDeleteCom**
SYNOPSIS:	PGS_PC_TempDeleteCom <logical ID>
DESCRIPTION:	This program will flag a temporary file as deleted in the PCF or shared memory at the command line.
INPUTS:	argc-number of command line arguments argv[0]-executable name (not processed but listed here anyway) argv[1]-logical ID of the temporary file to be deleted
OUTPUTS:	NONE
RETURNS:	PGS_S_SUCCESS PGS_SH_SYS_PARAM PGS_SH_PC_NODATA PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297

#  Delete the temporary file with the logical ID 12297
PGS_PC_TempDeleteCom $LogicalID
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

REQUIREMENTS: PGSTK-0521

Get File Size Command

NAME: **PGS_PC_GetFileSizeCom**

SYNOPSIS: PGS_PC_GetFileSizeCom <logical ID>

DESCRIPTION: This program will retrieve the file size of the file associated with the input logical ID and version in the users Process Control File (PCF).

INPUTS: argc-number of command line arguments
argv[0] - logical ID (in the PCF) of the desired file
argv[1] - file version number

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.  This example assumes there is an entry for
#  for a file in the users PCF with logical ID 101

LogicalID=101
Version=1

#  Get the physical file size associated with the user's
#  input arguments LogicalID and Version

SIZE= `PGS_PC_GetFileSizeCom $LogicalID $Version`
RETVAL=$?

#  Check the return value

if [ $RETVAL -eq 0 ]
then

#  SIZE now contains the file size.
#  continue normal processing...

            :
            :

else

#  handle error case...
            :
            :
fi
```

NOTES:	This program is designed to be run from within the PGE.
REQUIREMENTS:	PGSTK-1290

Toolkit Termination Command

NAME:	**PGS_PC_TermCom**
SYNOPSIS:	PGS_PC_TermCom <shared-memory-flag> <log-file-flag>
C:	N/A
FORTRAN:	N/A
DESCRIPTION:	This program runs the functions necessary to clean up shared memory, send runtime files, send logfiles, update the PCF, and remove temporary files.
INPUTS:	argc-number of command line arguments argv[0]-executable name (not processed but listed here anyway) argv[1]-flag stating whether or not to use shared memory argv[2]-flag stating whether or not to write to a log file
OUTPUTS:	NONE
RETURNS:	PGS_S_SUCCESS PGS_SH_PC_DELETETEMP PGS_SH_SMF_SENDRUNTIME PGS_SH_SMF_SENDLOGFILE PGS_SH_MEM_TERM

EXAMPLES:

```
PGS_PC_TermCom ShmOff LogOff
PGS_PC_TermCom ShmOn LogOff
```

NOTES:

The send file capability of PGS_PC_TermCom is SCF functionality. This functionality will be disabled at the Release B DAACs, but will remain available to the SCF toolkit.

The PGS_PC_TermCom tool was developed two years ago to allow SCF developers to send files to other locations in the absence of a data distribution capability. This toolkit tool was not meant to replace the ECS DAAC distribution system, but to supply functionality prior to the system availability. Instrument teams can use the distribution system, by writing an ESDT for QA files. The subscription service (B.1) can then push the files to the requestor.

In the B.0 timeframe, there is no push, per se. A work-around could be to use the Version 0 Client ordering function. Or, an email message could be sent, announcing the presence of a QA file. If this message were sent to a special account, a script could then be run to pull the QA files out of the DAAC. This is a temporary solution, prior to B.1 operation.

If a PGE Fails:. Files are marked for sending, packaged up in a Failed Production History tar file (if and only if the PGE fails), and archived on the Data Server. The SCF is then notified and can retrieve it. If the PGE succeeds, the marked files are not put into a tar file.

The SCF Functionality:

This program is designed to be run from within the PGS_PC_Shell.sh script and is not intended to be run as a stand alone program from the command line. Running this program outside the script PGS_PC_Shell.sh will give undefined results.

Since this tool now supports the transfer of status and runtime files, certain steps need to be performed by the user to ensure that this transfer operation is carried-out properly.

FILE TRANSFER SETUP

The current transfer mechanism (ftp) requires the use of a '.netrc' file, which must reside in the user's home directory on the execution host. 'ftp' accesses this file to establish a connection with the remote host. Once the connection is made, the process of performing the actual file transfer can proceed.

This file must contain information in the following format:

machine <hostname> login <username> password <userpassword>

For example:

machine adriatic login guest password anonymous

For reasons of security, the '.netrc' file should ONLY have read permission for the user, (i.e., -rw-------).

(Refer to the man pages on netrc for more information.)

PROCESS CONTROL SETUP

As part of the transfer operation, this tool also transmits a notification message to the interested parties to inform them as to the disposition of the requested runtime and status files.

As with many other Process Control tools, this tool depends on certain entries in the Process Control File. The values of these entries however are user defined according to their local environment.

Refer to the standard Process Control File to find the following entries:

10109|TransmitFlag; 1=transmit,0=disable|0
- Set to 1 to enable file/e-mail transmission.

10106|RemoteHost|<hostname>
- Host should be the same as that which appears in the '.netrc' file.

10107|RemotePath|<destination directory>
- Directory must be writeable and large enough to hold the transferred data.

10108|EmailAddresses|<list of notification addresses>
- Notification message indicates which files have been transferred and where they currently reside.

WARNING-Do not attempt to transfer files to the same host and directory that this program is running on. The original files will be deleted in accordance with the ftp protocol for sending and receiving files. That is to say that, upon determination that the destination file is the same as the source; the destination file will be removed before sending the source file.

REQUIREMENTS: PGSTK-1311

6.2.3.2 Process Control API Tools

Get a File Reference from Logical

NAME: **PGS_PC_GetReference( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetReference(
          PGSt_PC_Logical     prodID,
          PGSt_integer        *version,
          char                *referenceID)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_getreference(prodid,version,referenceid)
          integer        prodid
          integer        version
          character*200  referenceid
```

DESCRIPTION: This tool may be used to obtain a physical reference (file name) from a logical identifier.

INPUTS: prodID-User defined constant identifier that internally represents the current product.

version-Version of reference to get. Remember, for standard input files there can be a many-to-one relationship.

OUTPUTS: referenceID-The actual file reference returned as a string

version-The number of versions remaining for the requested Product ID

RETURNS:

Table 6-51. PGS_PC_GetReference Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_REFERENCE_FOUND	link number does not have the data that mode is requesting
PGSPC_E_DATA_ACCESS_ERROR	problem while accessing PCS data

EXAMPLES:

C:

```
#define           MODIS1A 2530

PGSt_integerversion;
char              referenceID[PGSd_PC_FILE_PATH_MAX];
PGSt_SMF_status   returnStatus;

/# Get first version of the file #/
version = 1;

returnStatus =
      PGS_PC_GetReference(MODIS1A,&version,referenceID);

/# version now contains the number of versions remaining #/

if (returnStatus != PGS_S_SUCCESS)
      goto EXCEPTION;
else
{ /# perform necessary operations on file #/ }
                .
                .
                .
EXCEPTION:
      return returnStatus;
```

FORTRAN:

```
implicit none

integer           version
character*135     referenceid
integer           returnstatus
integer           pgs_pc_getreference
integer           modis1a
parameter   (modis1a = 2530)

C     Get the first version of the file
      version = 1

      returnstatus = getreference(modis1a,version,referenceid)

      if (returnstatus .ne. pgs_s_success)
          goto 9999
      else

C         perform necessary operations on file
          .
          .
          .
9999  return
```

NOTES: All reference identifier strings are guaranteed to be no greater than PGSd_PC_FILE_PATH_MAX characters in length (see PGS_PC.h).

The version returns the number of files remaining for the product group. For example, if there are eight (8) versions of a file when the user requests version one (1) the value seven (7) is returned in version. When the user requests version two (2) the value six (6) is returned in version, etc. Therefore, it is not recommended to use version as a loop counter that is also passed into PGS_PC_GetReference().

REQUIREMENTS: PGSTK-1290

Access File Reference Type from PCF

NAME: **PGS_PC_GetReferenceType( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetReferenceType(
        PGSt_PC_Logical     identifier
        PGSt_integer        *type)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_PC.f'
include 'PGS_PC_9.f'

integer function pgs_pc_getreferencetype(identifier,type)
        integer identifier
        integer type
```

DESCRIPTION: This tool may be used to ascertain the type of file reference that is associated with a logical identifier within the science software.

INPUTS: identifier-The logical identifier as defined by the user. (This value must be mapped to an actual value via the PCF.)

OUTPUTS: type-Reference types that are defined in the PGS_PC header file. Possible values are:

PGSd_PC_INPUT_FILE_NAME
PGSd_PC_OUTPUT_FILE_NAME
PGSd_PC_TEMPORARY_FILE
PGSd_PC_INTERMEDIATE_INPUT
PGSd_PC_INTERMEDIATE_OUTPUT
PGSd_PC_SUPPORT_IN_NAME
PGSd_PC_SUPPORT_OUT_NAME

RETURNS:

Table 6-52. PGS_PC_GetReferenceType Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_FILES_FOR_ID	The Product ID does not contain a physical reference.
PGSPC_E_ENVIRONMENT_ERROR	Environment variable not set
PGSPC_E_DATA_ACCESS_ERROR	Error accessing Process Control Status data

EXAMPLES:

C:

```
#define     INSTR_SCRATCH_SPACE 2001

PGSt_SMF_status returnStatus;
PGSt_PC_Logical fileIdentifier;
PGSt_integerfileType;

fileIdentifier = INSTR_SCRATCH_SPACE;

/# getting the type attribute of a file #/

returnStatus =
      PGS_PC_GetReferenceType(fileIdentifier,&fileType);
if (returnStatus != PGS_S_SUCCESS)
{
    goto EXCEPTION;
}
else
{
    switch (fileType)
    {
    case PGSd_PC_INPUT_FILE_NAME:
    case PGSd_PC_OUTPUT_FILE_NAME:
    case PGSd_PC_SUPPORT_IN_NAME:
    case PGSd_PC_SUPPORT_OUT_NAME:
        /#
            open standard product or support file
        #/
        returnStatus = PGS_IO_Gen_Open( );
                .
                .
                .
        break;

    case PGSd_PC_INTERMEDIATE_INPUT:
    case PGSd_PC_INTERMEDIATE_OUTPUT:
    case PGSd_PC_TEMPORARY_FILE:
        /#
            open temporary or intermediate file
        #/
            returnStatus = PGS_IO_Gen_Temp_Open( );
                .
                .
                .
            break;
```

```
                    default:
                            /#
                                invalid type returned only in the event that
                                call to *GetReferenceType was not successful
                            #/

                    } /# end switch (fileType) #/
                }
                              .
                              .
                              .
                EXCEPTION:
                        return returnStatus;
```

FORTRAN:

```
                implicit none

                INTEGER INSTR_SCRATCH_SPACE
                PARAMETER (INSTR_SCRATCH_SPACE = 2001)

                integer returnstatus
                integer fileidentifier
                integer filetype
                integer pgs_pc_getreferencetype

                fileidentifier = INSTR_SCRATCH_SPACE

C            getting the type attribute of a file

                returnstatus =
                      pgs_pc_getreferencetype(fileidentifier,filetype)
                if (returnstatus .ne. pgs_s_success) then
                    goto 9999
                else if (
                        (filetype .eq. PGSd_PC_INPUT_FILE_NAME) .or.
                        (filetype .eq. PGSd_PC_OUTPUT_FILE_NAME) .or.
                        (filetype .eq. PGSd_PC_SUPPORT_IN_NAME) .or.
                        (filetype .eq. PGSd_PC_SUPPORT_OUT_NAME)
                        ) then

C            open standard product or support file

                        returnstatus = PGS_IO_Gen_OpenF(...);
                                .
                                .
                                .
                else if (
                        (filetype .eq. PGSd_PC_INTERMEDIATE_INPUT) .or.
                        (filetype .eq. PGSd_PC_INTERMEDIATE_OUTPUT) .or.
```

```
                          (filetype .eq. PGSd_PC_TEMPORARY_FILE)
                          ) then
C            open temporary or intermediate file
                          returnstatus = PGS_IO_Gen_Temp_OpenF(...);
                                  .
                                  .
                                  .
                  else
C            invalid type returned only in the event that
C            call to *GetReferenceType was not successful
endif
9999    return
```

NOTES: This tool will return the reference type (mode) for files that have references in a Process Control File (PCF). This tool will not identify runtime parameters as such.

In order for this tool to function properly, a valid Process Control File will need to be created first. Please refer to Appendix C (User's Guide) for instructions on how to create and validate such a file.

REQUIREMENTS: PGSTK-1290.

Generate a Unique ID

NAME: **PGS_PC_GenUniqueID( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GenUniqueID(
        PGSt_PC_Logical     prodID,
        char                *uniqueID)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_genuniqueid(prodid,uniqueid)
        integer         prodid
        character*200   uniqueid
```

DESCRIPTION: This tool may be used to generate a unique product identifier. This identifier may be attached to file metadata to facilitate tracking of production output. The identifier may include Production Run ID, the Science Software Program ID, and the actual Product ID.

INPUTS: prodID-The logical identifier as defined by the user. The user's definitions will be mapped into actual identifiers during the Integration & Test procedure.

OUTPUTS: uniqueID-The unique ID generated by this function. This ID will be returned as a string. The ID is guaranteed to be no greater than PGSd_PC_LABEL_SIZE_MAX in length (see PGS_PC.h).

RETURNS:

Table 6-53. PGS_PC_GenUniqueID Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data

EXAMPLES:

C:

```
#define           CERES3A 300

PGSt_SMF_status   returnStatus;
char              uniqueID[PGSd_PC_LABEL_SIZE_MAX];

returnStatus = PGS_PC_GenUniqueID(CERES3A,uniqueID);
```

```
                  if (returnStatus != PGS_S_SUCCESS)
                          goto EXCEPTION;
                  else
                  {
/# attach uniqueID into file metadata field #/
                  }
                                    .
                                    .
                                    .
                  EXCEPTION:
                           return returnStatus;
```

FORTRAN:

```
                  implicit none

                  integer           returnstatus
                  character*200     uniqueid
                  integer           pgs_pc_genuniqueid
                  integer           ceres3a
                  parameter         (ceres3a = 300)

                  returnstatus = pgs_pc_genuniqueid(ceres3a,uniqueid)

                  if (returnstatus .ne. pgs_s_success) then
                          goto 9999
                  else
C         attach uniqueid into file metadata field
                  endif
                          .
                          .
                          .
return
```

NOTES: If more than one product is being generated from the same PGE, then the appropriate product identifier must be used as input to this function when called from within the science software. Upon entry into this function all input values will be checked to determine that legal values were passed in. If any value is illegal, the function will return the proper error value to the calling function. All unique identifier strings are guaranteed to be no greater than PGSd_PC_LABEL_SIZE_MAX characters in length (see PGS_PC.h).

REQUIREMENTS: PGSTK-1280.

Get User Defined Configuration Values

NAME: **PGS_PC_GetConfigData( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetConfigData(
        PGSt_PC_Logical     configParamID,
        char                *configParamVal)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_getconfigdata(configparamid,
*                                     configparamval)
        integer        configparamid
        character*200  configparamval
```

DESCRIPTION: This tool may be used to import run-time configuration parameters into the PGE.

INPUTS: configParamID-User defined constant that internally represents a configuration parameter.

OUTPUTS: configParamVal-A string representation of the configuration parameter value. No interpretation of this value will be done in the Toolkit; the value returned will be left to the application programmer.

RETURNS:

Table 6-54. PGS_PC_GetConfigData Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_CONFIG_FOR_ID	no configuration data for product id
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data

EXAMPLES:

C:

```
#define           MODIS1A_CONFIG1 2990

char              configParamVal[PGSd_PC_VALUE_LENGTH_MAX];
PGSt_SMF_status   returnStatus;
long              config1;
```

```
returnStatus =
      PGS_PC_GetConfigData(MODIS1A_CONFIG1,configParamVal);

if (returnStatus != PGS_S_SUCCESS)
        goto EXCEPTION;
else
{
   /# MODIS1A_CONFIG1 is integral parameter #/
          config1 = atoi(configParamVal);

          if (config1 > 0)
          {
                  /# activate sub-process A #/
          }
          else
          {
                  /# activate sub-process B #/
          }
}
                  .
                  .
                  .

EXCEPTION:
        return returnStatus;
```

FORTRAN:

```
implicit none

character*200     configparamval
integer           returnstatus
integer           pgs_pc_getconfigdata
integer           config1
integer           modis1a_config1
parameter         (modis1a_config1 = 2990)

returnstatus =
      pgs_pc_getconfigdata(modis1a_config1,configparamval)

if (returnstatus .ne. success) then
        goto 9999
else
```

```
C
C                       modis1a_config1 is integral parameter
C                       assuming you have a function to convert character
C                       data to integer data - called.....strtoint.
C                       strtoint(configparamval,config1)
```

```
                          if (config1 .gt. 0) then
C                                 activate sub-process A
                          else
C                                 activate sub-process B
                          .
                          .
                          .
                          endif
endif
return
```

NOTES: All configuration parameter value strings are guaranteed to be less than PGSd_PC_VALUE_LENGTH_MAX characters in length (see PGS_PC.h). There will be a shell script command version of this routine to retrieve configuration information from the script.

REQUIREMENTS: PGSTK-1290.

Get Number of Files Associated with a Product

NAME: **PGS_PC_GetNumberOfFiles( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetNumberOfFiles(
        PGSt_PC_Logical     prodID,
        PGSt_integer        *numFiles)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_getnumberoffiles(prodid,numfiles)
        integer prodid,
        integer numfiles)
```

DESCRIPTION: This tool may be used to determine the number of files that are associated with a particular Product ID. A many-to-one relationship may exist with Product Input, Product Output Support Input and Support Output files. This function will give the user a way to determine how many files exist for a product ID.

INPUTS: prodID-The logical identifier as defined by the user. The user's definitions will be mapped into actual identifiers during the Integration & Test procedure.

OUTPUTS: numberOfFiles-Total number of files for a particular product ID.

RETURNS:

Table 6-55. PGS_PC_GetNumberOfFiles Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_FILES_FOR_ID	incorrect number of configuration parameters
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data

EXAMPLE:

C:

```
#define     CERES4 7090

PGSt_integernumFiles;
PGSt_integerversion;
```

```
                  PGSt_SMF_status   returnStatus;
                  int               loopCounter;
                  char              ceresFiles[10][PGSd_PC_FILE_PATH_MAX];

                  returnStatus = PGS_PC_GetNumberOfFiles(CERES4,&numFiles);

                  if (returnStatus != PGS_S_SUCCESS)
                          goto EXCEPTION;
                  else
                  {

/# loop and get file names #/

                        for (loopCounter = 0; loopCounter < numFiles;
                              loopCounter++)
                        {

/# specify which file to get #/

version = loopCounter + 1;

/# save references for future use #/

                              returnStatus =
                                    PGS_PC_GetReference(CERES4,&version,
                                          ceresFiles[loopCounter]);
                        }
                  }
                              .
                              .
                              .
                  EXCEPTION:
                         return returnStatus;
```

FORTRAN:

```
                  implicit none

                  integer           numfiles
                  integer           version
                  integer           returnstatus
                  integer           loopcounter
                  character*355     referenceid
                  character*355     ceresfiles(10)
                  integer           pgs_pc_getnumberoffiles
                  integer           pgs_pc_getreference
                  integer           ceres4
                  parameter         (ceres4 = 7090)

                  returnstatus = pgs_pc_getnumberoffiles(ceres4,numfiles)
```

```
       if (returnstatus .ne. pgs_s_success)
             goto 9999
       else
             do 100 loopcounter = 1,numfiles
                   version = loopcounter
                   returnstatus = pgs_pc_getreference(ceres4,
       *                                           version,
       *                                           ceresfiles(loopcounter))
100          continue
             .
             .
             .
9999   return
```

NOTES: This function will allow a one-to-many relationship to exist between logical and physical file name. The file version number is returned in reverse order. For example, if there are eight (8) versions of a Product ID and the user requests the first one, the value eight (8) would be returned in numFiles.

REQUIREMENTS: PGSTK-1290

Get the Attribute of the File Associated with the Particular Product ID and Version

NAME: **PGS_PC_GetFileAttr( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetFileAttr(
        PGSt_PC_Logical     prodID,
        PGSt_integer        version
        PGSt_integer        formatFlag,
        PGSt_integer        maxSize,
        char                *fileAttribute)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_getfileattr(prodid,version,formatflag,fileAttribute)
        integer prodid
        integer version
        integer formatflag
        integer maxSize
        character*(*)   fileAttribute
```

DESCRIPTION: This tool may be used to retrieve an attribute associated with a particular product ID and version number. The data placed in the attribute will be defined and interpreted by the user. The SDP Toolkit has no dependency on the attribute.

INPUTS: prodID-The logical identifier as defined by the user. The user's definitions will be mapped into actual identifiers during the Integration & Test procedure.

version-The particular version of the Product ID that the attribute is being requested from. With files there may be a many-to-one relationship.

formatFlag-Flag indicating method of attribute return. Possible values are:

PGSd_PC_ATTRIBUTE_LOCATION
PGSd_PC_ATTRIBUTE_STRING

maxSize-Amount of space allocated for attribute if formatFlag is PGSd_PC_ATTRIBUTE_STRING.

OUTPUTS: fileAttribute-The actual file attribute

If formatFlag is PGSd_PC_ATTRIBUTE_LOCATION then fileAttribute will return the file containing the attribute.

If formatFlag is PGSd_PC_ATTRIBUTE_STRING then fileAttribute will return the attribute as a string.

RETURNS:

Table 6-56. PGS_PC_GetFileAttr Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_REFERENCE_FOUND	no reference found matching product id and version number
PGSPC_W_ATTR_TRUNCATED	not enough space passed in for attribute
PGSPC_W_NO_ATTR_FOR_ID	a physical reference was found but no attribute exists for that reference
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data
PGSPC_E_INVALID_MODE	invalid format flag value passed in

EXAMPLE:

C:

```
#define            MODIS1A 4220

PGSt_integerversion;
PGSt_integermaxSize;
PGSt_SMF_status    returnStatus;
char               fileAttribute[PGSd_PC_FILE_PATH_MAX];

version = 1;
maxSize = 0;

/# get the attribute file name of the first MODIS1A file #/

returnStatus = PGS_PC_GetFileAttr(MODIS1A,version,
      PGSd_PC_ATTRIBUTE_LOCATION,maxSize,fileAttribute);

if (returnStatus != PGS_S_SUCCESS)
        goto EXCEPTION;
else
{

/# open attribute file and search attribute for particular
   data #/

}
              .
```

```
                    .
                    .
        EXCEPTION:
                return returnStatus;
```

FORTRAN:

```
        implicit none

        integer          version
        integer          returnstatus
        integer          maxsize
        character*355    fileattribute
        integer          pgs_pc_getfileattr
        integer          modis1a
        parameter        (modis1a = 4220)

        version = 1
        maxsize = 355

C         get the attribute file name of the first modis1a file

        returnstatus = pgs_pc_getfileattr(modis1a,version,
              PGSd_PC_ATTRIBUTE_LOCATION,maxsize,fileattribute)

        if (returnstatus .ne. pgs_s_success) then
                goto 9999

else

C                       open attribute file and search attribute for
C                       particular data

        endif
                .
                .
                .
        return
```

NOTES: Allocating enough space for the attribute variable will be the responsibility of the application programmer. This function will write the attribute into fileAttribute for maxSize bytes or the end of the attribute, which ever comes first.

REQUIREMENTS: PGSTK-1290, PGSTK-1310

Get the Version Number of the Particular File Matching the Attribute

NAME: **PGS_PC_GetFileByAttr( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetFileByAttr(
          PGSt_PC_Logical     prodID,
          PGSt_integer        (*searchFunc)(char *attr),
          PGSt_integer        maxSize,
          PGSt_integer        *version)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function
pgs_pc_getfilebyattr(prodid,searchfunc,
*                         maxsize,version)
          integer prodid
          integer searchfunc
          integer maxSize
          integer version
```

DESCRIPTION: This tool may be used to retrieve the version number associated with a file with a particular attribute.

INPUTS: prodID-The logical identifier as defined by the user. The user's definitions will be mapped into actual identifiers during the Integration & Test procedure.

searchFunc-A user defined function that performs the search on the attribute. This function must be passed in as a type PGSt_integer function. It should return type PGSd_PC_MATCH upon a successful attribute match or PGSd_PC_NO_MATCH upon an unsuccessful attribute match.

maxSize-Maximum amount of space to place into attribute.

OUTPUTS: version-The version number of the file with the successful attribute match

RETURNS:

Table 6-57. PGS_PC_GetFileByAttr Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_ATTR_MATCH	did not find a match with the specified product ID
PGSPC_W_NO_ATTR_FOR_ID	the product ID contains no attribute
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data

EXAMPLE:

C:

```
#define MODIS1A   5775

PGSt_integersearchfunc_(char *attr);   /# function
                                                prototype #/

/# The function passed into PGS_PC_GetFileByAttr() MUST be
   called #/
/# searchfunc_#/

PGSt_integermaxSize;
PGSt_integerversion;
PGSt_SMF_status   returnStatus;
char              referenceID[PGSd_PC_FILE_PATH_MAX];

maxSize = 300;

returnStatus = PGS_PC_GetFileByAttr(MODIS1A,searchfunc_,
                                    maxSize,&version);

if (returnStatus != PGS_S_SUCCESS)
        goto EXCEPTION;
else
{
/# get file reference #/

      returnStatus =
            PGS_PC_GetReference(MODIS1A,version,referencID);

}
              .
              .
              .
EXCEPTION:
        return returnStatus;
```

FORTRAN:

```
                  implicit none

                  integer     version
                  integer     searchfunc

C    The function passed into pgs_pc_getfilebyattr() MUST be called searchfunc

                  integer           maxsize
                  integer           returnstatus
                  integer           pgs_pc_getfilebyattr
                  integer           pgs_pc_getreference
                  character*355     referenceid
                  integer           modis1a
                  parameter         (modis1a = 5775)

                  maxsize = 300

                  returnstatus = pgs_pc_getfilebyattribute(modis1a,
                 *          searchfunc,maxsize,version)

                  if (returnstatus .ne. pgs_s_success) then
                          goto 9999
                  else

C
C                 get file reference
C
                          returnstatus = pgs_pc_getreference(modis1a,version,
             *                                   referenceid)
                  endif
                          .
                          .
                          .
                  return
```

NOTES: The attribute checking is left to the application programmer. The attribute for comparison must be passed into searchFunc by means of a global variable. The attribute to be compared against will be passed into searchFunc by the function PGS_PC_GetFileByAttr( ). The function searchFunc must have declared a variable large enough to handle the incoming attribute. The attribute will be read until maxSize bytes or end of file, which ever come first.

REQUIREMENTS: PGSTK-1290

Check Process Control Information File (PCF)

NAME:	**pccheck.sh**
SYNOPSIS:	pccheck.sh [-h] <-i user-PCF> [-o numbered-PCF] [-c standard PCF] [-s]
C:	N/A
FORTRAN	N/A

DESCRIPTION: The purpose of this tool is to assist the developer in setting up a Process Control File (PCF). This utility will help to point out simple syntax and content errors that might lead to more serious runtime errors, if left uncorrected. This tool will not, however, detect errors in logic, nor will it correct PCF files.

INPUTS: -i <PCF>-The -i flag will be followed by the Process Control Information File. This flag is mandatory.

- o <outfile>-The -o flag will be followed by a file name that will be output by this command. The name of output file must be a file that does not already exist. This flag is optional.
- h-Upon receiving the -h flag a short description of the usage of pccheck.sh will be provided to the user and the command will exit.
- c-The -c option will cause a compare to be run against a specified template file. The compare will only compare the reserved Product ID’s.
- s-The -s flag will cause all output except for the output from the -c flag to be suppressed.

OUTPUTS: NONE

RETURNS:

0	- Normal completion
1	- Error condition

EXAMPLE:

```
pccheck.sh -i $PGSHOME/runtime/pcf.fil -o out.fil
pccheck.sh -o out.fil -i $PGSHOME/runtime/pcf.fil
pccheck.sh -i $PGSHOME/runtime/pcf.fil -o out.fil -c
      $PGSRUN/PC/PCF.v3
pccheck.sh -i $PGSHOME/runtime/pcf.fil -c $PGSRUN/PC/PCF.v3
      -s
pccheck.sh -i in.fil
pccheck.sh -h
```

NOTES: This shell script accepts an input file (PCF) and an optional output file. The output file will be an exact copy of the input file except that line numbers are inserted into the file. This output file is provided as a convenience to the user when analyzing the generated report, which sometimes references line locations in the original PCF. This utility is also capable of comparing against a “standardized” PCF file to detect changes that have been made to the SDP Toolkit specific records (those with reserved logical identifiers in the 10K-11K range); the optional suppression flag prevents all output, other than the comparison results, from being reported.

REQUIREMENTS: PGSTK-1313

Get Universal Reference from Logical

NAME: **PGS_PC_GetUniversalRef()**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetUniversalRef(
        PGSt_PC_Logical     prodID,
        PGSt_integer*       version,
        char                *universalRef)
```

FORTRAN:

```
include ‘PGS_SMF.f’
include ‘PGS_PC.f’
include ‘PGS_PC_9.f’

integer function
pgs_pc_getuniversalref(prodid,version,universalref)
        integer prodid
        integer version
        character*150 universalref
```

DESCRIPTION: This tool may be used to obtain a universal reference from a logical identifier.

INPUTS: prodID-User defined constant identifier that internally represents the current product.

version-Version of reference to get. Remember, for Product Input files and Product Output files there can be a many-to-one relationship.

OUTPUTS: universalRef-The actual universal reference returned as a string.

RETURNS:

Table 6-58. PGS_PC_GetReference Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_REFERENCE_FOUND	link number does not have the data that mode is requesting
PGSPC_E_DATA_ACCESS_ERROR	problem while accessing PCS data
PGSPC_W_NO_UREF_DATA	the product id and version contains no universal reference data

EXAMPLES:

```
C:
                    #define     MODIS1A 2530

                    PGSt_integerversion;
                    char  universalRef[PGSd_PC_UREF_LENGTH_MAX];
                    PGSt_SMF_status   returnStatus;

                    /# Get first version of the file #/
                    version = 1;

                    returnStatus =
                    PGS_PC_GetUniversalRef(MODIS1A,version,universalRef);

                    if (returnStatus != PGS_S_SUCCESS)
                          goto EXCEPTION;
                    else
                    { /# perform necessary operations on file #/ }
                          .
                          .
                          .
                    EXCEPTION:
                          return returnStatus;


FORTRAN:            IMPLICIT NONE

                    integer     version
                    character*150     universalRef
                    integer     returnstatus
                    integer     pgs_pc_getuniversal
                    integer     modis1a
                    parameter   (modis1a = 2530)

C                   Get the first version of the file
                    version = 1

                    returnstatus = getreference(modis1a,version,referenceid)
                    if (returnstatus .ne. pgs_s_success)
                          goto 9999
                    else

C                   perform necessary operations on file
                          .
                          .
                          .
                    9999  return
```

NOTES:	All reference identifier strings are guaranteed to be no greater than PGSd_PC_UREF_LENGTH_MAX characters in length (see PGS_PC.h).
	The version returns the number of files remaining for the product group. For example, if there are eight (8) versions of a when the user requests version one (1) the value seven (7) is returned in version. When the user requests version two (2) the value six (6) is returned in version, etc. Therefore, it is not recommended to use version as a loop counter that is also into PGS_PC_GetReference().
REQUIREMENTS:	PGSTK-1290

Get Size of a File

NAME: **PGS_PC_GetFileSize()**

SYNOPSIS:

C:

```
#include <PGS_PC.h>
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_PC_GetFileSize(
        PGSt_PC_Logical     prodID,
        PGSt_integer        version,
        PGSt_integer*       filesize)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_getfilesize(prodid,version,filesize)
        integer prodid,
        integer version,
        integer filesize)
```

DESCRIPTION: This tool may be used to obtain the size of a file from a logical identifier.

INPUTS: prodID-The logical identifier as defined by the user.
version - Version of reference to get.

OUTPUTS: filesize - The size of a file.

RETURNS:

Table 6-59. PGS_PC_GetFileSize Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_REFERENCE_FOUND	link number does not have the data that mode is requesting
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data
PGS_E_UNIX	Unix system error
PGS_E_TOOLKIT	an unexpected error occurred

EXAMPLE:

C:

```
#define     PROD_ID 10501

PGSt_integerversion;
PGSt_integerfilesize;
PGSt_SMF_status   returnStatus;

/# Get first version of the file #/
version = 1;

returnStatus =
PGS_PC_GetFileSize(PROD_ID,version,&filesize);

/# version now contains the number of versions remaining #/

if (returnStatus != PGS_S_SUCCESS
goto EXCEPTION;

else

{ /# perform necessary operations on file #/ }

      .

      .

      .

EXCEPTION:

      return returnStatus;
```

FORTRAN:

NOTES: In order for this tool to function properly, a valid Process Control file will need to be created first. Please refer to Appendix C (User's Guide) for instructions on how to create such a file.

REQUIREMENTS: PGSTK-1290

6.2.4 Shared Memory Management Tools

The tools described in this section provide for a limited use of shared memory amongst executables within a PGE. These tools allow for the creation of a single user memory segment within a PGE, and for the subsequent attachment and detachment of that memory segment to another executable within the same PGE. Due to the way in which shared memory is accessed, the APIs for the C and FORTRAN programming languages are necessarily different. C users may directly manipulate the shared memory area but FORTRAN users are limited to copying to and from the shared memory area via intermediary Toolkit functions. **Note that the operation of these tools is contingent on the assumption that the user will make proper use of the initialization and termination commands that have been provided with this release of the Toolkit (please note that the Memory Management initialization and termination routines supplied with Toolkit 3 have been subsumed by corresponding Process Control commands that MUST be invoked before and after the execution of the PGE respectively). The shell utility PGS_PC_Shell.sh already activates the initialization and termination commands, so user activation of these commands should not be performed if the shell utility is used.**

Create Shared Memory Segment

NAME: **PGS_MEM_ShmCreate( )**

SYNOPSIS:

C:
```
#include <PGS_MEM1.h>

PGSt_SMF_status
PGS_MEM_ShmCreate(
        PGSt_uinteger size);
```

FORTRAN:
```
integer function pgs_mem_shmcreate(size)
integer size
```

DESCRIPTION: This tool may be used to create a shared memory segment. This tool should only be called once in a given processing script (PGE).

INPUTS size-size of the shared memory segment in bytes

OUTPUTS: None

RETURNS:

Table 6-60. PGS_MEM_ShmCreate Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_ENV	Environment Variable “PGSMEM_SHM_SYSKEY” is not set
PGSMEM_E_SHM_MAXSIZE	Maximum system-imposed shared memory exceeded
PGSMEM_E_SHM_MULTICREATE	More than one shared-memory is created for a given PGE

EXAMPLES:

C:
```
typedef struct
{
   int  id;
   char msg[100];
}TestStruct;

TestStruct  *shmPtr;
PGSt_SMF_status  returnStatus;

returnStatus = PGS_MEM_ShmCreate(sizeof(TestStruct);
if (returnStatus == PGS_S_SUCCESS)
{
   returnStatus = PGS_MEM_ShmAttach((void **)&shmPtr);
   if (returnStatus == PGS_S_SUCCESS)
   {
      shmPtr->id = 123;
      strcpy(shmPtr->msg,”Writing data into shared memory”);
   }
}
```

FORTRAN:

```
integer       pgs_mem_shmcreate

integer       returnstatus
integer       shm_size
character*100 test_string
shm_size = 100
test_string = “Writing data into shared memory”

returnstatus = pgs_mem_shmcreate(shm_size)
if (returnstatus .eq. pgs_s_success) then
    returnstatus = pgs_mem_shmwrite(test_string, shm_size)
endif

!  the contents of test_string have been written to shared
!  memory which can be accesses by another process in the
!  PGE
```

NOTES: This shared memory scheme is not A POSIX implementation and will therefore be subjected to change when the POSIX.4 implementation is available. System limitations will define the amount of memory that can be allocated as a shared-memory segment. Only one memory segment may be created per PGE; it may however be attached/detached as many times as are required.

REQUIREMENTS: PGSTK-1241

Attach Shared Memory Segment

NAME: **PGS_MEM_ShmAttach( )**

SYNOPSIS:

C:

```
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_MEM_ShmAttach(
        void **shm);
```

FORTRAN: None

DESCRIPTION: This tool may be used by an executable to attach to an existing shared memory segment. PGS_MEM_ShmCreate( ) should already be called, either within the same executable or from an earlier executable within the PGE. If the shared memory segment has been detached by calling PGS_MEM_ShmDetach( ), then you may re-attach the segment to your process-space again.

INPUTS: shm-pointer referencing the shared memory segment

OUTPUTS: shm-pointer referencing the shared memory segment

RETURNS:

Table 6-61. PGS_MEM_ShmAttach Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_ENV	Environment variable PGSMEM_SHM_SYSKEY is not set
PGSMEM_E_SHM_NOTCREATE	Shared-memory has not been attached to the process
PGSMEM_E_SHM_MULTIATTACH	Multiply attached shared-memory in a process

EXAMPLES:

```
typedef struct
{
   intid;
   char     msg[100];
}TestStruct;

PGSt_SMF_status  returnStatus;
TestStruct  *shmPtr;
```

PROCESS A:

```
returnStatus = PGS_MEM_ShmCreate(sizeof(TestStruct));
if (returnStatus == PGS_S_SUCCESS)
{
   returnStatus = PGS_MEM_ShmAttach((void **)&shmPtr);
   if (returnStatus == PGS_S_SUCCESS)
   {
      shmPtr->id = 123;
      strcpy(shmPtr->msg,”From Process A”);
   }
}
```

PROCESS B:

```
returnStatus = PGS_MEM_ShmAttach((void **)&shmPtr);
if (returnStatus == PGS_S_SUCCESS)
{
   if ((shmPtr->id = 123) && (strcmp(shmPtr->msg,”From
                              Process A”) == 0))
   {
      printf(“Reading data from Process A successful”);
   }
}
```

NOTES: Before using this function, PGS_MEM_ShmCreate( ) should have already be called, either within the same executable or from an earlier executable within the PGE. If the shared memory segment has been detached by calling PGS_MEM_ShmDetach( ), then you may re-attach the segment to your process-space again.

This tool lets the system select the memory location for your shared memory, thereby allowing the system to make the best possible use of its memory resources.

This tool is not part of POSIX and is subjected to change when the POSIX.4 implementation becomes available.

REQUIREMENTS: PGSTK-1241

Detach Shared Memory Segment

NAME: **PGS_MEM_ShmDetach( )**

SYNOPSIS:

C:

```
#include <PGS_MEM1.h>

PGSt_SMF_status
PGS_MEM_ShmDetach(
        void);
```

FORTRAN: None

DESCRIPTION: This tool may be used to detach a shared memory segment from a process that it has been attached to.

INPUTS: None

OUTPUTS: None

RETURNS:

Table 6-62. PGS_MEM_ShmDetach Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_NOTATTACH	Shared-memory has not been attached to the process

EXAMPLES:

```
typedef struct
{
   intid;
   char     msg[100];
}TestStruct;

PGSt_SMF_status  returnStatus;
TestStruct  *shmPtr;

returnStatus = PGS_MEM_ShmCreate(sizeof(TestStruct));
if (returnStatus == PGS_S_SUCCESS)
{
   returnStatus = PGS_MEM_ShmAttach((void **)&shmPtr);
   if (returnStatus == PGS_S_SUCCESS)
   {
      shmPtr->id = 123;
      strcpy(shmPtr->msg,”Writing data into shared memory”);
```

```
        PGS_MEM_ShmDetach();
    }
}
```

NOTES: Note that this tool is not part of POSIX and is subjected to change when the POSIX.4 implementation becomes available. This function will only detach the shared memory segment from the process. The shared memory segment will not be removed from the system by calling this tool; therefore one can re-attach it again.

REQUIREMENTS: PGSTK-1241

Read from Shared Memory Segment

NAME: **PGS_MEM_ShmRead( )**

SYNOPSIS:

C: None

FORTRAN:
```
include ‘PGS_SMF.f
include ‘PGS_MEM_9.f’

integer function pgs_mem_shmread(mem_ptr, size)
        integer         size
        character       mem_ptr(size)
```

DESCRIPTION: This function copies the contents of shared memory into a user allocated (may be dynamically or statically allocated) memory area. This function is meant to be used by FORTRAN (77/90) users who cannot take advantage of the C shared memory tools PGS_MEM_ShmAttach() and PGS_MEM_ShmDetach().

INPUTS:

Table 6-63. PGS_MEM_ShmRead Inputs

Name	Description
size	size (in bytes) of mem_ptr (see below)

OUTPUTS:

Table 6-64. PGS_MEM_ShmRead Outputs

Name	Description
mem_ptr	array or structure to which the contents of the shared memory area will be written

RETURNS:

Table 6-65. PGS_MEM_ShmRead Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_ENV	Environment variable PGSMEM_SHM_SYSKEY is not set
PGSMEM_E_SHM_NOTCREATE	User defined shared-memory has not been created
PGSMEM_E_SHM_MULTIATTACH	Multiply attached shared-memory in a process
PGSMEM_E_SHM_NOTATTACH	Failed to attach shared memory to this process shared-memory

EXAMPLES:

FORTRAN:

```
                integer    pgs_mem_shmread
                integer    size

                character  shm_buffer(1000)

                integer    returnstatus

                returnstatus = pgs_mem_shmread(shm_buffer, size)

                if (returnstatus .ne. pgs_s_success) goto 999

                !  the contents of shared memory (which may contain data
                !  from a previous process) have been copied to shm_buffer

            999 continue   ! process error conditions
```

NOTES: This tool is meant to be used by FORTRAN (77/90) users ONLY. C users should use the functions PGS_MEM_ShmAttach() and PGS_MEM_ShmDetach().

The tool PGS_MEM_ShmCreate() MUST be called before PGS_MEM_ShmRead() is invoked.

This tool is not part of POSIX and is subjected to change when the POSIX.4 implementation becomes available.

The user passes in a pointer to a user defined memory area (an area of memory which has been either statically or dynamically allocated by the user) and the size of that area. This function will retrieve the pointer to the shared memory area and copy the contents of the shared memory into the users memory area. This function will then detach the shared memory from the current process. Before exiting from the PGE, the system will make sure that the attached shared memory segment will be removed from the system.

REQUIREMENTS: PGSTK-1241

Write to Share Memory Segment

NAME: **PGS_MEM_ShmWrite( )**

SYNOPSIS:

C: None

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_MEM_9.f'

integer function pgs_mem_shmwrite(mem_ptr, size)
integer         size
character       mem_ptr(size)
```

DESCRIPTION: This function copies the contents of a user allocated (may be dynamically or statically allocated) memory area into shared memory. This function is meant to be used by FORTRAN (77/90) users who cannot take advantage of the C shared memory tool PGS_MEM_ShmAttach() and PGS_MEM_ShmDetach().

INPUTS:

Table 6-66. PGS_MEM_ShmWrite Inputs

Name	Description
mem_ptr	array or structure the contents of which will be written to the shared memory area
size	size (in bytes) of mem_ptr (see above)

OUTPUTS: NONE

RETURNS:

Table 6-67. PGS_MEM_ShmWrite Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_ENV	Environment variable PGSMEM_SHM_SYSKEY is not set
PGSMEM_E_SHM_NOTCREATE	User defined shared-memory has not been created
PGSMEM_E_SHM_MULTIATTACH	Multiply attached shared-memory in a process
PGSMEM_E_SHM_NOTATTACH	Failed to attach shared memory to this process shared-memory

EXAMPLES:

FORTRAN:

```
                integer    pgs_mem_shmwrite

                integer    size
                integer    returnstatus

                character  shm_buffer(1000)

                !  fill shm_buffer with interesting data

                returnstatus = pgs_mem_shmwrite(shm_buffer, size)

                if (returnstatus .ne. pgs_s_success)  goto 999

                !  the contents of shm_buffer have been written to the
                !  shared memory area which can be accessed by a subsequent
                !  process

         999    continue   ! process error conditions
```

NOTES: This tool is meant to be used by FORTRAN (77/90) users ONLY. C users should use the functions PGS_MEM_ShmAttach() and PGS_MEM_ShmDetach().

The tool PGS_MEM_ShmCreate() MUST be called before PGS_MEM_ShmWrite() is invoked.

This tool is not part of POSIX and is subjected to change when the POSIX.4 implementation becomes available.

The user passes in a pointer to a user defined memory area (an area of memory which has been either statically or dynamically allocated by the user) and the size of that area. This function will retrieve the pointer to the shared memory area and write the contents of the users memory area to the shared memory area OVERWRITING whatever was previously in the shared memory area. This function will then detach the shared memory from the current process. Before exiting from the PGE, the system will make sure that the attached shared memory segment will be removed from the system.

REQUIREMENTS: PGSTK-1241

6.2.5 Bit Manipulation Tools

It is assumed that bit-manipulation functionality will be provided inherently by the language for ‘C’ and Fortran90 and that users of Fortran77 will use compilers that conform to MIL STD 1753 to obtain these capabilities.

6.2.6 Spacecraft Ephemeris and Attitude Data Access Tools

This tool group contains tools and associated software that provides access to the spacecraft ephemeris and attitude at a given time. Currently the EOS_AM, EOS_PM and TRMM platforms are supported. In this release of the Toolkit, orbit and attitude data is supplied by the ECS Spacecraft Orbit and Attitude Simulator.

6.2.6.1 Orbit and Attitude Simulator

The ECS Spacecraft Orbit and Attitude Simulator is based on Upper Atmosphere Research Satellite (UARS) FORTRAN code. It has been completely rewritten in C and revised for EOS.

6.2.6.1.1 Brief Description

The spacecraft orbit simulator *orbsim* will create files of simulated spacecraft orbit and attitude data necessary to test the SDP Toolkit spacecraft ephemeris and attitude data access tool (PGS_EPH_EphemAttit( )) in the SCF environment. Users may alternatively create their own data files but MUST follow the ECS ephemeris and attitude file formats.

WARNING: this simulator uses a relatively simple algorithm and is meant to produce data for software testing ONLY. This data should not be used for any actual processing or for prediction purposes.

6.2.6.1.2 The SCF Environment

At the DAACs the users will be responsible for submitting the criteria upon which ephemeris and attitude files will be staged for their PGE. The DAACs will populate the Process Control File (PCF) appropriately based on this user supplied criteria. In the SCF environment users must populate the PCF with appropriate ephemeris and attitude data files themselves. No tools that require access to spacecraft ephemeris data will function without these ephemeris and attitude files. An ephemeris file and an attitude file must be provided for any time during which processing will be requested.

The PCF file provided with the Toolkit contains the Logical IDs which have been reserved for the ephemeris and attitude data files. There is one Logical ID for each type of data and the appropriate Logical ID MUST be used for each set of ephemeris and attitude files. Replace the dummy values in the PCF with the actual location of the ephemeris and attitude files to be used. Use the given ephemeris file Logical ID for all ephemeris data files and the given attitude file Logical ID for all attitude files. To include multiple files of either type use file versioning. The order of the files is not important, the ephemeris and attitude access tool will sort the files before attempting to access them (WARNING: providing files with overlapping start/stop times may produce unexpected results).

The unconfigured ephemeris and attitude Logical ID entries in the PCF look as follows (respectively):

```
10501|INSERT_EPHEMERIS_FILES_HERE|||||1
10502|INSERT_ATTITUDE_FILES_HERE|||||1
```

The configured entries should look something like this:

```
10501|TRMM_1994-01-12.eph|~/database/sun5/EPH||||3
10501|TRMM_1994-01-13.eph|~/database/sun5/EPH||||2
10501|TRMM_1994-01-14.eph|~/database/sun5/EPH||||1
10502|TRMM_1994-01-12.att|~/database/sun5/EPH||||3
10502|TRMM_1994-01-13.att|~/database/sun5/EPH||||2
10502|TRMM_1994-01-14.att|~/database/sun5/EPH||||1
```

See Section 6.2.3 Process Control Tools for a discussion of the PCF and file versioning.

6.2.6.1.3 Running the Orbit/Attitude Simulator

The executable *orbsim* is installed in the $PGSBIN directory at installation time. Make sure the $PGSBIN directory is in your path. To run the program, type “orbsim” at the command line prompt (from any directory).

The simulator is self-explanatory (if you read the messages on the screen). A “q” may be entered at any prompt to quit the simulator. At most prompts there will be a default value that can be selected by merely returning at the prompt without typing any characters. These default values will be indicated by “[]” (e.g., enter a number [7]:).

The first prompt will request the spacecraft ID. The supported values for this are: TRMM, EOS_AM, EOS_PM.

The second prompt will request the start day. Enter the start day in CCSDS ASCII time code (format A or B-see Time and Date Conversion Tools). Only the “date” portion of this input will be used, any “time” portion will be ignored. The third prompt will request the stop day that should be entered using the same format as the start day. The start and stop days are inclusive (e.g., entering the same start and stop days will create the spacecraft ephemeris file for that day). The fourth prompt will request the data (or time) interval in seconds. This number is a real number that represents the time interval between data records in the file. These times represent actual ephemeris data. This data will be returned to users directly through PGS_EPH_EphemAttit( ). Ephemeris data requested at times other than the actual record times will be interpolated. After reading in the time interval, the simulator will display the start and stop day and time interval entered, as well as the total size (in megabytes) of the data files that will be created. The simulator will then request confirmation of these input values. If the values are rejected the simulator will request the information again beginning with the start day until the values are accepted.

Once the time information has been entered and confirmed the simulator will issue a prompt requesting attitude “noise”. This simulator does not allow for any specific yaw, pitch or roll variation, however attitude noise may be introduced to simulate small random variations in the yaw, pitch and roll data reported. At the noise prompt the maximum desired amplitude in arcseconds of the noise should be entered. This should be entered as a real number whose magnitude is LESS than 1000.0 arcseconds (only the magnitude will be considered; the sign of the number will be ignored). The next prompt will be for attitude rate noise. This should be entered as a real number whose magnitude is LESS than 1000.0 arcseconds/second. Entering “N” at the first prompt (for attitude noise) will turn off this feature; and the roll, pitch and yaw will always be reported as exactly zero. No noise is the default behavior.

The simulator will then prompt for the directory where the ephemeris and attitude files it generates should be written to. The default installation directory is determined from the location of the file leapsec.dat which is assumed to be in $PGSDAT/TD, the simulator will then define the default directory as $PGSDAT/EPH. The location of the output directory is not significant to the tool PGS_EPH_EphemAttit() in any way. The simulator will issue a prompt indicating the default location and asking that the installation directory be specified. Any valid directory may be specified at this prompt (a relative path may be used). The default directory can be selected by merely entering return at this prompt. If an invalid directory is entered the prompt will be reissued until a valid directory is entered.

After a valid directory has been indicated the simulator will attempt to create the spacecraft ephemeris and attitude files for the times requested. The simulator will generate one file each of ephemeris data and attitude data for each date specified. The files generated will follow the naming convention <sc_name>_<date>.eph and <sc_name>_<date>.att for ephemeris and attitude files respectively. The file names and lengths generated by the simulator are for convenience only. Ephemeris and attitude data files may actually have any name and be of any time duration. However, because of the simulator convention of one ephemeris file and one attitude file per day, the simulator will NOT overwrite an existing file for the same spacecraft and the same day, an error message will be issued and the file(s) will be skipped. If for any other reason a file cannot be created the simulator will issue an error message and a prompt asking whether or not it should continue. If directed to continue, the simulator will try one more time to create the file and then continue on to the next file without further warning whether or not the file could be created. The most likely scenario for this is when the user does not have write permission for the directory specified. The above mentioned prompt allows the user to change the directory permission and continue. If the simulator is unable to write to a file that it has already opened (e.g., the disk is full) an error message will be issued.

When all files requested have been written (or skipped), a final prompt is issued allowing the whole process to be repeated.

6.2.6.1.4 Spacecraft Ephemeris And Attitude File Formats

See Appendix L (ECS Spacecraft Ephemeris and Attitude File Formats)

6.2.6.1.5 Tools that Require Spacecraft Ephemeris Files

PGS_EPH_EphemAttit( )
PGS_EPH_GetEphMet( )
PGS_CBP_body_inFOV( )
PGS_CBP_Sat_CB_Vector( )
PGS_CSC_GetFOV_Pixel( )

PGS_CSC_SubSatPoint( )
PGS_CSC_Earthpt_FOV( )
PGS_CSC_Earthpt_FixedFOV( )
PGS_CSC_ECItoORB( )
PGS_CSC_ORBtoECI( )
PGS_CSC_ECItoSC( )
PGS_CSC_SCtoECI( )
PGS_CSC_ORBtoSC( )
PGS_CSC_SCtoORB( )

6.2.6.1.6 Warning

The files created by the simulator can be very large and keeping many of them around can quickly fill a hard drive (one day of orbit data for EOS_AM at the default time interval is nearly nine megabytes). The size of the files can be reduced by choosing larger time intervals between data records.

This tool will create files for time in the far future or distant past if the user specifies them. The time of each record in spacecraft ephemeris and attitude files is kept in SDP Toolkit internal time (see Time and Date Conversion Tools) which is a form of TAI time. The user will not be notified if the file created is outside the times for which TAI is defined or currently known (relative to a corresponding UTC time). The simulator will estimate the time and create the file.

6.2.6.2 Ephemeris File Checker

The ECS Spacecraft Ephemeris File Checker can be used to check the format of exiting spacecraft ephemeris files and/or attitude files. This is useful for verifying that an ephemeris file or an attitude file created by a user (i.e., not using the ECS Spacecraft Orbit and Attitude Simulator) is properly formatted. The Ephemeris File Checker is also useful in checking on the time resolution and spacecraft ID of an existing spacecraft ephemeris file or attitude file.

6.2.6.2.1 Brief Description

The spacecraft ephemeris file checker (chkeph) will check the contents of spacecraft ephemeris and attitude files. The checker will read the file header and verify that the metadata contained therein is reasonable. If the header checks out, the checker will then check each record in the file to verify that the times are properly specified (i.e., that the records are properly spaced in time).

6.2.6.2.2 Running the Ephemeris File Checker

The executable *chkeph* is installed in the $PGSBIN directory at installation time. Make sure the $PGSBIN directory is in your path. To run the program type “chkeph” at the prompt with the name(s) of any file(s) to be checked, e.g.,

```
chkeph TRMM_1998-02-01.eph TRMM_1998-02-02.eph
```

If the file to be checked is not in the same directory as the one from which chkeph was invoked, the path name must be specified as well (e.g., chkeph ../EPH/TRMM_1998-02-02.eph).

For each file specified chkeph will print out the data contained in the header and check the data records. The first line printed will be the name of the spacecraft and the corresponding numeric value of the Toolkit spacecraft ID (if the spacecraft is an ECS supported s/c). The next two lines will be the numeric start and stop times (respectively) indicated in the header in internal time. Each time will be followed on the same line with the CCSDS ASCII Code (format A) representation of the equivalent UTC time. The next line will be the time interval. Note that this quantity is for record keeping only (i.e., the value has no effect on Toolkit operation). Users creating their own files (i.e., without using the orbsim utility--see above) may set this field to any value. The next line will be the number of records expected to be in the file based on the number of records specified in the file header. The first record will be checked to verify that the time of the record is the same as the time specified as the start time in the file header. Each subsequent record will then be checked to verify that the time of the record is greater than the time of the record immediately preceding it. The last record in the file will be checked to verify that the time of the record is the same as the time specified as the stop time in the file header. The Ephemeris File Checker will issue appropriate error messages if it finds anomalies in the contents of the file that it is checking.

6.2.6.3 Spacecraft Tags Definition File

As of Toolkit 5.2, spacecraft tags are no longer “hard-coded”. Spacecraft tags are defined in an ASCII data file and looked up at runtime. This allows the Toolkit geolocation tools to effectively support any spacecraft that has had it’s ephemeris and attitude data formatted for the Toolkit (see Appendix L. Ephemeris And Attitude File Formats). The spacecraft tags definition file is referenced via the Process Control File with the logical ID of 10801. The file contains a series of records (one per line) of the form:

<sc_tag>,<sc_name>,<eao>

Where:

<sc_tag>	is the numerical (integer) value of the spacecraft tag (passed to Toolkit functions).
<sc_name>	is the actual name of the spacecraft as contained in the ephemeris/attitude file header (see Tables L-1 and L-5).
<eao>	is a string consisting of three digits describing the order of the Euler angles (e.g.: 321, 312, 212) as contained in the attitude file (see Table L-5).

As delivered the Tookit is configured to support the TRMM, EOS-AM1 and EOS-PM platforms. These entries in the spacecraft tags file should not be altered. Additional entries may be added below these entries. Each entry should have a unique <sc_name> and <sc_tag>.

To ensure backward compatibility, the previous implementation of spacecraft tags has been retained in the Toolkit software. That is, if the tag is TRMM, EOS-AM1, or EOS-PM and the Spacecraft Tags Definition File is not found, the Toolkit will execute the old “hard coded’ method.

Get Ephemeris and Attitude

NAME: **PGS_EPH_EphemAttit( )**

SYNOPSIS:

C:

```
#include <PGS_EPH.h>

pgst_SMF_status
PGS_EPH_EphemAttit(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_boolean        orbFlag,
        PGSt_boolean        attFlag,
        PGSt_integer        qualityFlags[][2],
        PGSt_double         positionECI[][3],
        PGSt_double         velocityECI[][3],
        PGSt_double         eulerAngles[][3],
        PGSt_double         xyzRotRates[][3],
        PGSt_double         attitQuat[][4])
```

FORTRAN:

```
include ‘PGS_SMF.f’
include ‘PGS_TD.f’
include ‘PGS_TD_3.f’
include ‘PGS_EPH_5.f’

integer function pgs_eph_ephemattit(spacecrafttag,numvalues,asciiutc,
                offsets,orbflag,attflag,qualityflags,
                positioneci,velocityeci,eulerangles,
                xyzrotrates,attitquat)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             orbflag
        integer             attflag
        integer             qualityflags(2,*)
        double precision    positioneci(3,*)
        double precision    velocityeci(3,*)
        double precision    eulerAngles(3,*)
        double precision    xyzrotrates(3,*)
        double precision    attitquat(4,*)
```

DESCRIPTION: This tool gets ephemeris and/or attitude data for the specified spacecraft at the specified times.

INPUTS:

Table 6-68. PGS_EPH_EphemAttit Inputs

Name	Description	Units	Min	Max
spacecraftTag	spacecraft identifier	N/A		
numValues	num. of values requested	N/A		
asciiUTC	UTC time reference start time in CCSDS ASCII time code A format	ASCII	1961-01-01	see NOTES
offsets	array of time offsets in seconds relative to asciiUTC	seconds	depends on asciiUTC	
orbFlag	set to true to get ephemeris data	T/F		
attFlag	set to true to get attitude data	T/F		

OUTPUTS:

Table 6-69. PGS_EPH_EphemAttit Outputs

Name	Description	Units
qualityFlags	quality flags for position and attitude data	see NOTES
positionECI	ECI position	meters
velocityECI	ECI velocity	meters/sec
eulerAngles	s/c attitude as a set of Euler angles	radians
xyzRotRates	angular rates about body x, y and z axes	radians/sec
attitQuat	spacecraft to ECI rotation quaternion	N/A

RETURNS:

Table 6-70. PGS_EPH_EphemAttit Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSEPH_W_BAD_EPHEM_VALUE	One or more values could not be determined
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephemeris/attitude files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephemeris/attitude files could be found for input times
PGSEPH_E_NO_DATA_REQUESTED	Both orbit and attitude flags are set to false
PGSTD_E_SC_TAG_UNKNOWN	Unrecognized/unsupported spacecraft tag
PGSEPH_E_BAD_ARRAY_SIZE	Array size specified is less than 0
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for initial time (asciiUTC)
PGS_E_TOOLKIT	An unexpected error occured

EXAMPLES:

C:

```
#define ARRAY_SIZE 10

PGSt_double       offsets[ARRAY_SIZE];
PGSt_double       positionECI[ARRAY_SIZE][3];
PGSt_double       velocityECI[ARRAY_SIZE][3];
PGSt_double       eulerAngles[ARRAY_SIZE][3];
PGSt_double       xyzRotRates[ARRAY_SIZE][3];
PGSt_double       attitQuat[ARRAY_SIZE][4];

char              asciiUTC[28];

PGSt_integerqualityFlags[ARRAY_SIZE][2];

int               i;

PGSt_SMF_status   returnStatus;

** initialize asciiUTC and offsets array **

strcpy(asciiUTC,"1998-02-03T19:23:45.123");
for (i=0;i<ARRAY_SIZE;i++)
  offsets[i] = (PGSt_double) i;

returnStatus = PGS_EPH_EphemAttit(PGSd_EOS_AM, numValues,
                  asciiUTC, offsets, PGS_TRUE, PGS_TRUE,
                  qualityFlags, positionECI, velocityECI,
                  eulerAngles, xyzRoteRates, attitQuat);

if (returnStatus != PGS_S_SUCCESS)
{
            :
** do some error handling **
            :
}
```

FORTRAN:

```
integer           numvalues/10/
integer           i
integer           returnstatus
integer           qualityflags(2,numvalues)

character*27asciiutc

double precision  offsets(numvalues)
double precision  positioneci(3,numvalues)
double precision  velocityeci(3,numvalues)
double precision  eulerangles(3,numvalues)
double precision  xyzrotrates(3,numvalues)
double precision  attitquat(4,numvalues)

C              initialize asciiutc and offsets array
```

```
      asciiutc = '1998-02-03T19:23:45.123'
      do 100 i = 1,numvalues
100     offsets(i) = i-1
      returnstatus =  pgs_eph_ephemattit(pgsd_eos_am,numvalues,
      >                  asciiutc,pgs_true,
      >                  pgs_true,attflag,
      >                  qualityflags,positioneci,
      >                  velocityeci,eulerangles,
      >                  xyzroterates,attitquat)
      if (returnstatus .ne. pgs_s_success) then
                :
      *** do some error handling ***
                :
      endif
```

NOTES: **The Euler angles are always relative to the geocentrically based orbital reference frame The attitude rates for TRMM are relative to geodetic orbital reference. The attitude rates for AM1 and later spacecraft are relative to inertial (J2000) reference. In all cases, the attitude rates are the spacecraft angular velocity vector projected on the body axes.**

QUALITY FLAGS:

The quality flags are returned as integer quantities but should be interpreted as bit fields. Only the first 32 bits of each quality flag is meaningfully defined, any additional bits should be ignored (currently integer quantities are 32 bits on most UNIX platforms, but this is not guaranteed to be the case—e.g. an integer is 64 bits on a Cray).

Generally the quality flags are platform specific and are not defined by the Toolkit. Two bits of these flags have, however, been reserved for SDP Toolkit usage. Bit 12 will be set by the Toolkit if no data is available at a requested time, bit 14 will be set by the Toolkit if the data at the requested time has been interpolated (the least significant bit is "bit 0"). Any other bits are platform specific and are the responsibility of the user to interpret. See also Section L.3 (Quality Flags).

See Section 6.2.7.1 (Time Acronyms)

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Appendix L (ECS Spacecraft Ephemeris and Attitude File Formats)

TIME OFFSETS:

This function accepts an ASCII UTC time, an array of time offsets and the number of offsets as input. Each element in the offset array is an offset in seconds relative to the initial input ASCII UTC time.

An error will be returned if the number of offsets specified is less than zero. If the number of offsets specified is actually zero, the offsets array will be ignored. In this case the input ASCII UTC time will be converted to Toolkit internal time (TAI) and this time will be used to process the data. If the number of offsets specified is one (1) or greater, the input ASCII UTC time will be converted to TAI and each element 'i' of the input data will be processed at the time: (initial time) + (offset[i]).

Examples:

if numValues is 0 and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0), then input[0] will be processed at time 432000.0 and return output[0]

if numValues is 1 and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0), then input[0] will be processed at time 432000.0 + offsets[0] and return output[0]

if numValues is N and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0), then each input[i] will be processed at time 432000.0 + offsets[i] and the result will be output[i], where i is on the interval [0,N) ([1,N] in the case of FORTRAN)

ERROR HANDLING:

This function processes data over an array of times (specified by an input ASCII UTC time and an array of time offsets relative to that time).

If processing at each input time is successful the return status of this function will be PGS_S_SUCCESS (status level of 'S').

If processing at ALL input times was unsuccessful the status level of the return status of this function will be 'E'.

If processing at some (but not all) input times was unsuccessful the status level (see SMF) of the return status of this function will be 'W' AND all high precision real number (C: PGSt_double, FORTRAN: DOUBLE PRECISION) output variables that correspond to the times for which processing was NOT successful will be set to the value: PGSd_GEO_ERROR_VALUE. In this case users may (should) loop through the output testing any one of the aforementioned output variables against the value PGSd_GEO_ERROR_VALUE. This indicates that there was an error in processing at the corresponding input time and no useful output data was produced for that time.

Note: A return status with a status of level of ‘W’ does not necessarily mean that some of the data could not be processed. The ‘W’ level may indicate a general condition that the user may need to be aware of but that did not prohibit processing. For example, if an Earth ellipsoid model is required, but the user supplied value is undefined, the WGS84 model will be used, and processing will continue normally, except that the return status will be have a status level of ‘W’ to alert the user that the default earth model was used and not the one specified by the user. The reporting of such general warnings takes precedence over the generic warning (see RETURNS above) that processing was not successful at some of the requested times. Therefore in the case of any return status of level ‘W,’ the returned value of a high precision real variable generally should be examined for errors at each time offset, as specified above.

Special Note: for this tool, the associated quality flags will also indicate that no data is available for those points that could not be successfully processed (see QUALITY FLAGS above).

REQUIREMENTS: PGSTK-0720, PGSTK-0141

Get Ephemeris and Attitude Metadata

NAME: **PGS_EPH_GetEphMet( )**

SYNOPSIS:

C:

```
#include <PGS_EPH.h>

pgst_SMF_status
PGS_EPH_EphMet(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_integer*       numOrbits,
        PGSt_integer        orbitNumber[],
        char                orbitAscendTime[][28],
        char                orbitDescendTime[][28],
        PGSt_double         orbitDownLongitude[])
```

FORTRAN:

```
include ‘PGS_SMF.f’
include ‘PGS_TD.f’
include ‘PGS_TD_3.f’
include ‘PGS_EPH_5.f’

integer function pgs_eph_ephemattit(spacecrafttag,numvalues,asciiutc,
                offsets,numorbits,orbitnumber,orbitascendtime,
                orbitdescendtime,orbitdownlongitude)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             numorbits
        integer             orbitnumber(*)
        character*27        orbitascendtime(*)
        character*27        orbitdescendtime(*)
        double precision    orbitdownlongitude(*)
```

DESCRIPTION: This tool returns the metadata associated with toolkit spacecraft ephemeris/attitude files.

INPUTS:

Table 6-71. PGS_EPH_GetEphMet Inputs

Name	Description	Units	Min	Max
spacecraftTag	spacecraft identifier	N/A		
numValues	num. of values requested	N/A		
asciiUTC	UTC time reference start time in CCSDS ASCII time code A format	ASCII	1961-01-01	see NOTES
offsets	array of time offsets in seconds relative to asciiUTC	seconds	depends on asciiUTC	

OUTPUTS:

Table 6-72. PGS_EPH_GetEphMet Outputs

Name	Description	Units
numOrbits	number of orbits spanned by data set	N/A
orbitNumber	array of orbit numbers spanned by data set	N/A
orbitAscendTime	array of times of spacecraft northward equator crossings	ASCII
orbitDescedTime	array of times of spacecraft southward equator crossings	ASCII
orbitDownLongitude	array of longitudes of spacecraft southward equator crossings	radians

RETURNS:

Table 6-73. PGS_EPH_GetEphMet Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephemeris/attitude files could be found for input times
PGSEPH_E_EPH_BAD_ARRAY_VALUE	Array size specified is less than 0
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_SC_TAG_UNKNOWN	Unrecognized/unsupported spacecraft tag
PGS_E_TOOLKIT	An unexpected error occured

EXAMPLES:

C:

```
#include <PGS_EPH.h>


#define ORBIT_ARRAY_SIZE 5    /* maximum number of orbits
                                 expected */

#define EPHEM_ARRAY_SIZE 100  /* number of ephemeris data
                                 points */
```

```
PGSt_double     offsets[EPHEM_ARRAY_SIZE];
PGSt_double     orbitdownlongitude[ORBIT_ARRAY_SIZE][3];

PGSt_integer    numOrbits;

char            asciiUTC[28];
char            orbitAscendTime[ORBIT_ARRAY_SIZE][28];
char            orbitDescendTime[ORBIT_ARRAY_SIZE][28];

/* initialize asciiUTC and offsets array with the times for
   actual ephemeris records that will be processed (i.e. by
   some other tool) */

strcpy(asciiUTC,”1998-02-03T19:23:45.123”);

for (i=0;i<EPHEM_ARRAY_SIZE;i++)
{
    offsets[i] = (PGSt_double) i*60.0;
}

/* get the ephemeris metadata associated with these times */

returnStatus = PGS_EPH_GetEphMet(PGSd_EOS_AM,
                                 EPHEM_ARRAY_SIZE, asciiUTC,
                                 offsets,&numOrbits,
                                 orbitAscendTime,
                                 orbitDescendTime,
                                 orbitDownLongitude);

if (returnStatus != PGS_S_SUCCESS)
{
             :
** do some error handling **
             :
}

/* numOrbits will now contain the number of orbits spanned
   by the data set (as defined by asciiUTC and
   EPHEM_ARRAY_SIZE offsets). orbitAscendTime will contain
   numOrbits ASCII UTC times representing the time of
   northward equator crossing of the spacecraft for each
   respective orbit.  orbitDescendTime will similarly
   contain the southward equator crossing times and
   orbitDownLongitude will contain the southward equator
   crossing longitudes */
```

FORTRAN:

```
        implicit none

        include ‘PGS_EPH_5.f’
        include ‘PGS_TD.f’
        include ‘PGS_TD_3.f’
        include ‘PGS_SMF.f’

        integer orbit_array_size/5/   ! max. num. orbits expected
        integer ephem_array_size/100/ ! num. of ephem. data points

        double precision offsets(ephem_array_size)
        double precision orbitdownlongitude(orbit_array_size)(3)

        integer          numorbits

        character*27     asciiutc
        character*27     orbitascendtime(orbit_array_size)
        character*27     orbitdescendtime(orbit_array_size)

!           initialize asciiutc and offsets array with the times for actual

!           ephemeris records that will be processed (i.e. by some other tool)


        asciiutc = ‘1998-02-03t19:23:45.123’

        do 100 i=1,ephem_array_size

        offsets(i) = i*60.D0

100  continue

!           get the ephemeris metadata associated with these times


        returnStatus = pgs_eph_getephmet(pgsd_eos_am,
        >                                ephem_array_size, asciiutc,
        >                                offsets,numorbits,
        >                                orbitascendtime,
        >                                orbitdescendtime,
        >                                orbitdownlongitude)

        if (returnStatus .ne. pgs_s_success) then

                   :
        ** do some error handling **
                   :

        endif
```

```
!    numOrbits will now contain the number of orbits spanned by the data set
!    (as defined by asciiUTC and EPHEM_ARRAY_SIZE offsets). orbitAscendTime
!    will contain numOrbits ASCII UTC times representing the time of northward
!    equator crossing of the spacecraft for each respective orbit.
!    orbitDescendTime will similarly contain the southward equator crossing
!    times and orbitDownLongitude will contain the southward equator crossing
!    longitudes
```

NOTES: see NOTES section of PGS_EPH_EphemAttit( )

REQUIREMENTS: PGSTK-0720, PGSTK-0141

6.2.6.3 EPH Functions

PGS_EPH_EphemAttit

See description in 6.2.6.3 Spacecraft Ephemeris and Attitude Tool.

PGS_EPH_GetEphMet

See description in 6.2.6.3 Get Ephemeris and Attitude Metadata.

PGS_EPH_interpolateAttitude

Given a pair of spacecraft attitudes (as Euler angles), attitude rates and their corresponding times this function interpolates the spacecraft attitude and attitude rates to a requested time between the two input times.

PGS_EPH_interpolatePosVel

Given a pair of spacecraft position vectors, velocity vectors and their corresponding times this function interpolates the spacecraft position and veloctiy to a requested time between the two input times.

6.2.7 Time and Date Conversion Tools

The ability to convert easily and accurately between different representations of time is crucial to EOS science data processing. The time and date conversion routines in the SDP Toolkit will convert between spacecraft time, UTC, International Atomic Time (TAI) and Julian date, as well as converting double precision values to and from CCSDS ASCII formats. Time values are converted for use in science software and as parameters when performing geo-coordinate transformations. In addition, converting time parameters to ASCII or to other more easily read formats facilitates the time values being added to metadata and to various processing logs in a human-readable form.

The spacecraft, UTC, Julian Date, and other times used as input and output for the time and date conversion routines will be in accord with the Consultative Committee for Space Data Systems (CCSDS) standard time code formats where applicable. The formats are described in CCSDS Blue Book, Issue 2, *Time Code Formats*, (CCSDS 301.0-B-2) issued by the Consultative Committee for Space Data Systems (NASA Code- OS, NASA, Washington DC 20546), April 1990. Various EOS supported spacecraft will deliver time data in various CCSDS binary codes. The Toolkit will translate times from these codes to more user friendly formats. Therefore, binary formats will not be described in the present manual. The reader is referred to the Blue Book and to interface documents for the particular spacecraft of interest. The ASCII codes will be described herein both for the convenience of users, and because we have exercised discretion in permitting or forbidding certain truncations.

Because UTC as a real variable is discontinuous at leap seconds boundaries (approximately every one to two years) it has been decided to carry it only in ASCII formats. TAI time runs at the same (Standard International compatible) rate and will be carried as a double precision number, in two ways: Julian Date and seconds from Jan. 1, 1993 UTC midnight.

Toolkit times are either character strings (CCSDS ASCII format), an array of two high precision real values (Toolkit Julian Dates) or a single high precision real value (all other values).

6.2.7.1 Time Acronyms

GAST	Greenwich Apparent Sidereal Time
GMST	Greenwich Mean Sidereal Time
GPS	Global Positioning System
MJD	Modified Julian Date
TAI	International Atomic Time
TDB	Barycentric Dynamical Time
TDT	Terrestrial Dynamical Time
TJD	Truncated Julian Date
UT1	Universal Time
UTC	Coordinated Universal Time

6.2.7.2 ASCII Time Formats

The CCSDS ASCII Time Codes (A and B formats) are defined in the CCSDS Blue Book, pages 2-6 to 2-8. The full format requires all the subfields be present, but certain subsets of the complete time codes are allowed (pages 2-7 to 2-8 of the Blue Book). The Toolkit will handle input and output with slightly different restrictions.

CCSDS ASCII Time Code A as implemented by the Toolkit:

YYYY-MM-DDThh:mm:ss.d->dZ

[Example 2002-02-23T11:04:57.987654Z]

where

YYYY = a four character subfield for year, with value in range 0001-9999

MM = a two character subfield for month with values 01-12, leading zeros required

DD = a two character subfield for day with values in the range 01-eom, where eom is 28, 29, 30, or 31 according to the month (and, for February, the year)

The “T”, a separator, must follow the DD subfield; if and only if there are more characters after the DD subfield; the string will be accepted and parsed such that mm, ss, and d are treated as 0. In that case, a “Z” will still be accepted, but not required, at the end.

hh = a two character subfield for hours, with values 00-23

mm = a two character subfield for minutes, with values 00-59

ss = a two character subfield for seconds, with values 00-59 (00-60 in a positive leap second interval, 00-58 in the case of a negative leap second)

d->d an n-character subfield, ($n < 7$ for input $n = 6$ for output), for decimal fraction of a second, with each digit in the range 0-9. If the decimal point appears on input, digits must follow it.

Z - terminator, optional on input

The CCSDS ASCII Time Code B format, described on p. 2-7 of the Blue Book, is:

YYYY-DDDThh:mm:ss.d->dZ

[Example 2002-054T11:04:57.987654Z]

The format is identical to the Code A except that the month, day combination MM-DD is replaced by day of year, i.e.,

DDD = Day of Year as a 3 character subfield with values 001-365 in non leap years and 001-366 in leap years.

NOTE: The CCSDS Formats require all leading zeros be present.

ASCII Time Input

ASCII time input strings may be in either CCSDS ASCII Time Code A format or CCSDS ASCII Time Code B format. All Toolkit functions requiring input ASCII time strings will correctly identify either format.

The Toolkit requires input ASCII time strings to include at least full dates (in format A or B) and will accept ASCII time strings that include times with up to six digits after the decimal point, or subsets truncated from the right (i.e., fractions of a second, whole seconds, minutes, or hours can be omitted by the user and the values will be set to zero. If a subfield is omitted the whole subfield should be omitted; e.g., "ss" cannot be replaced by "s" for seconds.) The time string may also not end with a field delimiter: "T",":" or ".". Users are warned that no error status or message will issue if any of these subfields is missing, so long as truncation is from the right; users should be careful to pass a string of sufficient length to accommodate their data! The Toolkit will not accept truncations from the left; i.e., the year, month and day must be present as four, two, and two digits respectively, or the year as four digits and the day of year as three. Truncation from the left would be too dangerous in view of the coming century change.

Finally, the Toolkit will provide an error message, which will include passing one or more of the offending characters, if the format is violated by input data. In this context, day numbers in excess of the allowable value for the month (and year, for February) are considered errors in format (e.g., a fatal message will issue if DDD = 366 (format B) or MM = 02 and DD = 29 (format A) in a non leap year). A fatal message will issue if the integer part of the seconds subfield runs over 58 in the presence of a negative leap second or over 59 in the absence of a positive leap second. There is no protection against missing data in the presence of a positive leap second if the integer seconds subfield fails to read 60; in that case Toolkit routines cannot populate the leap second interval.

ASCII Time Output

All ASCII time output strings will be in CCSDS ASCII Time Code A format (except for the output of PGS_TD_ASCIItime_AtoB(), which will be in CCSDS ASCII Time Code B format).

The Toolkit will output the full format (date and time), to six digits in the fractional seconds, even though the accuracy may be poorer than one microsecond. There are two reasons why the Toolkit will output microseconds, even though most users will not want numbers more accurate than one millisecond: (i) At least one platform (AM1) plans to provide microseconds; we do not wish to degrade their resolution. (ii) We wish to provide for upgradeability.

The Toolkit will issue a terminal "Z" on the output string to facilitate identification of the end of string and to signify Universal time.

The output strings will be 27 characters in Code A, including the "Z", and 25 in Code B, including the "Z" (Note: this does NOT include the terminating NULL character required in C strings).

6.2.7.3 Toolkit Internal Time (TAI)

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to in the Toolkit as TAI (upon which it is based). Values are maintained as single high precision real numbers (C: PGSt_double, FORTRAN: DOUBLE PRECISION). The numbers will be negative until midnight, UTC Jan. 1, 1993 and positive after that. The whole number part carries whole seconds and the fractional part carries fractions of a second.

6.2.7.4 Toolkit Julian Dates

6.2.7.4.1 Format

Toolkit Julian dates are kept as an array of two real high precision numbers (C: PGSt_double, FORTRAN: DOUBLE PRECISION). The first element of the array should be the half integer Julian day (e.g., N.5 where N is a Julian day number). The second element of the array should be a real number greater than or equal to zero AND less than one (1.0) representing the time of the current day (as a fraction of that (86400 second) day. This format allows relatively simple translation to calendar days (since the Julian days begin at noon of the corresponding calendar day). Users of the Toolkit are encouraged to adhere to this format to maintain high accuracy (one number to track significant digits to the left of the decimal and one number to track significant digits to the right of the decimal). Toolkit functions that do NOT require a Julian type date as an input and that do return a Julian date will return it in the above mentioned format. Toolkit functions that require a Julian date as an input and do NOT return a Julian date will first convert (internally) the input date to the above format if necessary. Toolkit functions that have a Julian date as both an input and an output will assume the input is in the above described format but will not check and the format of the output may not be what is expected if any other format is used for the input.

6.2.7.4.2 Meaning

Toolkit “Julian dates” are all derived from UTC Julian Dates. A Julian date in any other time stream (e.g., TAI, TDT, UT1, etc.) is the UTC Julian date plus the known difference of the other stream from UTC (differences range in magnitude from 0 seconds to about a minute). Note that although UTC days having leap seconds actually contain 86401 seconds, this is not true for Julian Days of any kind as implemented in the Toolkit. TAI, UT1, TDT and TDB Julian Days are all 86400 seconds, while the UTC Julian Day with the leap second contains duplicate values for one second; only in ASCII form does it have 86401 distinct seconds.

6.2.7.4.3 Examples

In the following examples, all Julian Dates are expressed in Toolkit standard form as two double precision numbers. For display here, the two members of the array are enclosed in braces {} and separated by a comma.

a. UTC to TAI Julian dates conversion

The Toolkit UTC Julian date for 1994-02-01T12:00:00 is: {2449384.50, 0.5}. TAI-UTC at 1994-02-01T12:00:00 is 28 seconds (.00032407407407 days). The Toolkit TAI Julian date for 1994-02-01T12:00:00 is:

{2449384.50, 0.5 + .00032407407407} = {2449384.50, 0.50032407407407}

Note that the Julian day numbers in UTC and the target time stream may be different by + or - 1 for times near midnight.

b. UTC to UT1 Julian dates conversion

The Toolkit UTC Julian date for 1994-04-10T00:00:00 is: {2449452.50, 0.0}. UT1-UTC at 1994-04-10T00:00:00 is -.04296 seconds (-0.00000049722221 days). The Toolkit UT1 Julian date for 1994-04-10T00:00:00 is:

{2449452.50, 0.0 - 0.0000004972222}
= {2449452.50, -0.0000004972222}
= {2449451.50, 0.9999995027778}

6.2.7.5 Time Boundaries

Many of the Toolkit functions that require time as an input or output keep track of time in the SDP Toolkit internal time format (see above). Most of these functions depend on the file leapsec.dat that contains the values of TAI-UTC (leap seconds).

Some Toolkit functions also (or instead) rely on the file utcpole.dat that contains the values of UT1-UTC.

The times that can be processed by a function may depend on the values maintained in one or both of these files which are updated periodically with new values.

6.2.7.5.1 TAI-UTC Boundaries

The minimum and maximum times that can successfully be processed by functions requiring the value TAI-UTC depend on the file leapsec.dat that relates leap second (TAI-UTC) values to UTC Julian dates.. The file leapsec.dat contains dates of new leap seconds and the total leap seconds times on and after Jan 1, 1972. For times between Jan 1, 1961 and Jan 1, 1972 it contains coefficients for an approximation supplied by the International Earth Rotation Service (IERS) and the United States Naval Observatory (USNO). These approximations are the same as adopted by the Jet Propulsion Laboratory (JPL) ephemeris group that produces the DE series of solar system ephemerides, such as DE200, and are used consistently with IERS/USNO/JPL usage. For times after Jan 1, 1961, but before the last date in the file, the Toolkit sets TAI-UTC equal to the total number of leap seconds to date, (or to the USNO/IERS approximation, for dates before Jan 1, 1972). If an input date is before Jan 1, 1961 the Toolkit sets the leap seconds value to 0.0. This is consistent with the fact that, for civil timekeeping since 1972, UTC replaces Greenwich Mean Solar Time (GMT), which had no leap seconds. Thus for times before Jan 1, 1961, the user can, for most Toolkit-related purposes, encode Greenwich Mean Solar Time as if it were UTC. If an input date is after the last date in the file, or after Jan 1, 1961, but the file cannot be read, the function will use a calculated value of TAI-UTC based on a linear fit of the data known to be in the table as of early 1997. This value is a crude estimate and may be off by as much as 1.0 or more seconds. If the data file, leapsec.dat, cannot be opened, or the time is outside the range from Jan 1, 1961 to the last date in the file, the return status level will be 'E'. Even when the status level is 'E', processing will continue, using the default value of TAI-UTC (0.0 for times before Jan 1, 1961, or the linear fit for later times). Thus, the user should always carefully check the return status. For times between 1961 and 1972, the leap seconds file contains data used in approximations designed to correct Greenwhich Mean Time to as close an equivalent of UT1 as possible; the Toolkit thus determines Earth rotation from GMT in that period.

6.2.7.5.2 UT1-UTC Boundaries

UT1 is the standard measure of axial Earth rotation and is used by all Toolkit functions for geolocation that locate the spacecraft relative to Earth, or Earth relative to sky (inertial space). UT1 can be reversibly transformed to "Greenwich Hour Angle". It is therefore important to maintain accurate values of UT1. The minimum and maximum times that can successfully be processed by functions requiring the value UT1-UTC depend on the file utcpole.dat that relates UT1-UTC values to UTC dates. The file utcpole.dat starts at June 30, 1972.

The file utcpole.dat, which is maintained periodically, contains final (definitive) and predicted values for UT1 - UTC and related variables that describe polar motion, a small correction (~< 15 meters) to geographic positions due to polar wander and wobble. When the file is updated, the definitive data will reach to within a week in the past of the update time, and the predicted data will extend about one year into the future. A success status message will be returned if all input times correspond to final values. A warning status message will be returned if predicted values are encountered. An error message will be returned if the time requested is beyond the end of the predictions, or the file cannot be read. The "predicted" values are expected to be satisfactory for most users for several weeks, even if the file is not updated weekly as it should be, because the predictions are rather good for many weeks. Users who desire to reprocess for better accuracy (< 1 m Earth position) will notice their results changing. Because the U.S. Naval Observatory (USNO) gradually refines its older solutions for Earth rotation, which are captured in our file "utcpole.dat", changes at the millimeter to centimeter level may be noticed weeks later even for data processed with "final" values for UT1. The following Table, based on error estimates in the USNO data table “finals.data” as of April 23, 1996, indicates the one-sigma errors to be expected in using the file “utcpole.dat” . The days in the left column should be interpreted as days since the last update of the file. The error is due to the inability to predict Earth rotation precisely. The error for times in the recent past (not shown) is only of order < 10 cm. The "interim" data quality supported in TK5 is no longer carried. The first few weeks of predictions are as good as the old "interim" values. Note that the rather small error values in Table 6-62 are a tiny part of the overall difference, UT1 - UTC, which is typically in the range ~ -0.9 to 0.9 seconds, or ~ -420 to +420 m.

Table 6-74. Estimated Errors in UT1 Predictions (Milliseconds of Time and Equivalent Meters of Geolocation Error)

Prediction Period (Days)	Error (milliseconds) (1 std deviation)	Error (meters at the equator) (1 std deviation)
1	0.3	0.14
30	3.9	1.7
60	6.5	3.0
90	8.8	4.0
120	10.9	4.9
150	12.9	5.8
180	14.8	6.7
225	17.5	7.9
270	20.1	9.0
315	22.5	10.1
360	24.9	11.0
365	25.7	11.5

Because of the reduced accuracy with predicted UT1, and the maximum extension of one year to the predictions, when a relevant function is used, the should carefully check the return status. A success ('S') level status message will be returned if all input times correspond to final values. A warning ('W') level status message will be returned if any input times correspond to predicted values, even though the error may not be large enough to concern most users. An error ('E') level status message will be returned if the file utcpole.dat cannot be found or if an input time is outside the range of values in the file.

These error messages due to end-of-data could cause problems for users who wish to run simulations one year or more in advance. Users needing to run simulations in the far future can follow procedures shown on the Toolkit Home Page under "Upgrading to Toolkit 5.2" at their own risk. These procedures are risky in an SCF environment or other non-DAAC environment, because of the possibility of pointing at the edited (and hence, false) data files when processing real data. There could also be risk at a DAAC environment if anyone found a way to point at these files with an altered PCF, e.g. if a command-line run were possible in processing science data

6.2.7.6 Updating the Leap Seconds File

The file $PGSDAT/TD/leapsec.dat contains- leap seconds data, used by many tools. Since the information changes with time, the file must be periodically updated. The SDP Toolkit contains utilities to perform this update function.

The shell script **update_leapsec.sh**, which is found in $PGSBIN, will update the leapsec.dat file to the current date. The clearcase version, **update_leapsec_CC.sh**, will do the same job within a

Clear Case (CM) view. To maintain a current leapsec.dat, the appropriate script must be run at least every six months. Leap seconds are usually added at the start of January or July, and announced six months ahead. Therefore, mid January and mid July are appropriate times to update the file. There would be no harm in running update_leapsec.sh monthly, however, and this would allow for the possibility--unused so far--that leap seconds could be introduced at the start of April or October. Update_leapsec.sh calls PGS_TD_NewLeap, a C program that performs most of the actual update work.

The update is done by collecting the latest information via ftp from the U. S. Naval Observatory, which maintains a current list of leap seconds. (The leap seconds are actually declared about six months ahead by International Earth Rotation Service (IERS) in France. At the DAACs, the process is done automatically by the scheduler. At Science Computing Facilities, users running the script must have a ".netrc" file in their home directory with the entry

machine maia.usno.navy.mil login anonymous password EOSuser

The permissions on this file must be set to private (use "chmod go-rwx .netrc").

6.2.7.7 Time and Date Conversion Tools

Convert UTC to TAI Time

NAME: **PGS_TD_UTCtoTAI( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoTAI(
          char          asciiUTC[28],
          PGSt_double   *secTAI93);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_utctotai(asciiutc, sectai93)
          character*27            asciiutc
          double precision        sectai93
```

DESCRIPTION: This tool converts UTC time in CCSDS ASCII Time Code (A or B format) to Toolkit internal time (real continuous seconds since 12AM UTC 1-1-93).

INPUTS:

Table 6-75. PGS_TD_UTCtoTAI Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A format or ASCII Time Code B format	time	1961-01-01T00:00:00Z	see NOTES

OUTPUTS:

Table 6-76. PGS_TD_UTCtoTAI Outputs

Name	Description	Units	Min	Max
secTAI93	continuous seconds since 12AM UTC Jan. 1, 1993	seconds	-1009843225.5	see NOTES

RETURNS:

Table 6-77. PGS_TD_UTCtoTAI Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSTD_E_TIME_FMT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGS_E_TOOKIT	Something unexpected happened, execution aborted

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28];
PGSt_double       secTAI93;

strcpy(asciiUTC,”1993-01-02T00:00:00.000000Z”);
returnStatus = PGS_TD_UTCtoTAI(asciiUTC,&secTAI93);
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
         :
         :
}
```

```
printf(“TAI: %f\n”,secTAI93);
```

FORTRAN:

```
implicit none

integer           pgs_td_utctotai
integer           returnstatus
character*27asciiutc
double precision  sectai93

asciiutc = ‘1993-01-02T00:00:00.000000Z’
returnstatus = pgs_td_utctotai(asciiutc,sectai93)
if (returnstatus .ne. pgs_s_success) goto 999
write(6,*) ‘TAI: ‘, sectai93
```

NOTES: **TIME ACRONYMS:**

TAI is: International Atomic Time
UTC is: Universal Coordinated Time

TIME BOUNDARIES:

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

TOOLKIT INTERNAL TIME (TAI):

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to in the toolkit as TAI (upon which it is based).

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

REQUIREMENTS: PGSTK-1170, PGSTK-1210, PGSTK-1220

Convert TAI to UTC Time

NAME: **PGS_TD_TAItoUTC( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_TAItoUTC(
        PGSt_double   secTAI93,
        char          asciiUTC[28]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_taitoutc(sectai93, asciiutc)
        character*27          asciiutc
        double precision      sectai93
```

DESCRIPTION: This tool converts Toolkit internal time (real continuous seconds since 12AM UTC 1-1-93) to UTC time in CCSDS ASCII Time Code A format.

INPUTS:

Table 6-78. PGS_TD_TAItoUTC Inputs

Name	Description	Units	Min	Max
secTAI93	continuous seconds since 12AM UTC Jan. 1, 1993	seconds	-1009843225.577182	see NOTES

OUTPUTS:

Table 6-79. PGS_TD_TAItoUTC Outputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A format	time	1961-01-01T00:00:00	see NOTES

RETURNS:

Table 6-80. PGS_TD_TAItoUTC Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGS_E_TOOLKIT	Something radically wrong occurred

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       secTAI93;
char              asciiUTC[28];

secTAI93 = 86400.;
returnStatus = PGS_TD_TAItoUTC(secTAI93,asciiUTC);
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
          :
          :
}

printf(“UTC: %s\n”,asciiUTC);
```

FORTRAN:

```
implicit none

integer           pgs_td_taitoutc
integer           returnstatus
double precision  sectai93
character*27asciiutc

sectai93 = 86400.D0
returnstatus = pgs_td_taitoutc(sectai93,asciiutc)
if (returnstatus .ne. pgs_s_success) goto 999
write(6,*) ‘UTC: ‘, asciiutc
```

NOTES: **TIME ACRONYMS:**

TAI is: International Atomic Time
UTC is: Universal Coordinated Time

TIME BOUNDARIES:

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

TOOLKIT INTERNAL TIME (TAI):

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to in the toolkit as TAI (upon which it is based).

REFERENCES FOR TIME:

CCSDS 2301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

REQUIREMENTS: PGSTK-1170, PGSTK-1210, PGSTK-1220

Convert Toolkit Internal Time to TAI Julian Date

NAME: **PGS_TD_TAItoTAIjd()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>
PGSt_double *
PGS_TD_TAItoTAIjd(
        PGSt_double  secTAI93,
        PGSt_double  jdTAI[2])
```

FORTRAN

```
include "PGS_SMF.f"
include "PGS_TD_3.f"
double precision function pgs_td_taitotaijd(sectai93, jdtai)
double precision sectai93
double precision jdtai(2)
```

DESCRIPTION: This function converts time in TAI seconds since 12 AM UTC 1-1-1993 to TAI Julian date.

INPUTS:

Table 6-81. PGS_TD_TAItoTAIjd.c Inputs

Name	Description	Units	Min	Max
secTAI93	Toolkit internal time (seconds since 12 AM	seconds	UTC 1-1-1993	

OUTPUTS:

Table 6-82. PGS_TD_TAItoTAIjd Outputs

Name	Description	Units	Min	Max
jdTAI	TAI Julian date	days	2437300.5	see NOTES

RETURNS: TAI Julian date (address of jdTAI).

EXAMPLES:

C:

```
PGSt_double      secTAI93;
PGSt_double      jdTAI[2];
secTAI93 = 86400.;
```

```
PGS_TD_TAItoTAIjd(secTAI93,jdTAI);

** jdTAI[0] should now have the value: 2448989.5 **
** jdTAI[1] should now have the value: 0.0003125 **
```

FORTRAN:

```
double precision sectai93

double precision jdtai

sectai93 = 86400.D0

call pgs_td_taitotaijd(sectai93, taijd)

! jdtai[0] should now have the value: 2448989.5

! jdtai[1] should now have the value: 0.0003125
```

NOTES: TAI is: International Atomic Time

The translation to and from UTC begins Jan 1, 1961. It is valid until about 6 months after the last leap second, in $PGSDAT/TD/leapsec.dat. When the script $PGSBIN/TD/update_leapsec.sh is run regularly the leap seconds file will be kept current and will be valid six months ahead.

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems)

Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK - 1220, 1160, 1170

Convert TAI Julian Date to Toolkit Internal Time

NAME: **PGS_TD_TAIjdtoTAI()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>
PGSt_double
PGS_TD_TAIjdtoTAI(
        PGSt_double  jdTAI[2])
```

FORTRAN:

```
double precision function pgs_td_taijdtotai(jdtai)
double precision jdtai(2)
```

DESCRIPTION: This function converts TAI Julian date to time in TAI seconds since 12 AM UTC 1-1-1993.

INPUTS:

Table 6-83. PGS_TD_TAIjdtoTAI Inputs

Name	Description	Units	Min	Max
jdTAI	TAI Julian date	days	2437300.5	ANY

OUTPUTS: None

RETURNS: Toolkit internal time (seconds since 12 AM UTC 1-1-1993).

EXAMPLES:

```
C     PGSt_double      secTAI93;
      PGSt_double      jdTAI[2];


      jdTAI[0] = 2448989.5;
      jdTAI[1] = 0.0003125;


      secTAI93 = PGS_TD_TAIjdtoTAI(jdTAI);
      /* secTAI93 should now have the value: 86400.*/
```

NOTES: TAI is: International Atomic Time

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems)

Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK - 1220, 1160, 1170

Convert TAI to GAST

NAME: **PGS_TD_TAItoGAST( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_TAItoGAST(
        PGSt_double  secTAI93,
        PGSt_double  *gast)
```

FORTRAN:

```
include‘PGS_SMF.f’
include‘PGS_CSC_4.f
include‘PGS_TD_3.f’

integer function pgs_td_taitogast(sectai93,gast)
        double precision        sectai93
        double precision        gast
```

DESCRIPTION: This function converts TAI (toolkit internal time) to Greenwich Apparent Sidereal Time (GAST) expressed as the hour angle of the true vernal equinox of date at the Greenwich meridian (in radians).

INPUTS:

Table 6-84. PGS_TD_TAItoGAST Inputs

Name	Description	Units	Min	Max
secTAI93	continuous seconds since 12AM UTC Jan. 1, 1993	seconds	-426297609.0	see NOTES

OUTPUTS:

Table 6-85. PGS_TD_TAItoGAST Outputs

Name	Description	Units	Min	Max
gast	Greenwich Apparent Sidereal Time	radians	0	2PI

RETURNS:

Table 6-86. PGS_TD_TAItoGAST Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_W_PREDICTED_UT1	Status of UT1-UTC correction is predicted
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSTD_E_NO_UT1_VALUE	No UT1-UTC correction available
PGS_E_TOOLKIT	Something radically wrong occured

EXAMPLES: None

NOTES: **TIME ACRONYMS:**

GAST is: Greenwich Apparent Sidereal Time
TAI is: International Atomic Time

TOOLKIT INTERNAL TIME (TAI):

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to in the toolkit as TAI (upon which it is based).See Section 6.2.7.4 Time and Date Conversion Tool Notes

TIME BOUNDARIES:

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCES FOR TIME:CCSDS 2301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

REQUIREMENTS: PGSTK-1170, PGSTK-1210

Convert UTC Time to Spacecraft Clock Time

NAME: **PGS_TD_UTC_to_SCtime( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTC_to_SCtime(
          PGSt_tag       spacecraftTag,
          char           asciiUTC[28],
          PGSt_scTime    scTime[8]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD.f'
include'PGS_TD_3.f'

integer function pgs_td_utc_to_sctime(spacecrafttag, asciiutc, sctime)
          integer        spacecrafttag
          character*27   asciiutc
          character*8    sctime
```

DESCRIPTION: This tool converts UTC in CCSDS Time Code A or B to spacecraft clock time in platform dependent format.

INPUTS: spacecraftTag-Spacecraft identifier; must be one of: PGSd_TRMM, PGSd_EOS_AM, PGSd_EOS_PM

asciiUTC-UTC time in CCSDS ASCII Time Code A or CCSDS ASCII Time Code B format. The values of MAX, and MIN depend on the spacecraft, see the files containing the specific conversions for more information

OUTPUTS: scTime-Spacecraft clock time in platform dependent CCSDS format. UNITS, MAX, and MIN depend on the spacecraft, see the files containing the specific conversions for more information.

RETURNS:

Table 6-87. PGS_TD_UTCtoSCtime Returns

Return	Description
PGS_S_SUCCESS	Successful execution
PGSTD_E_SC_TAG_UNKNOWN	Unknown spacecraft tag
PGSTD_E_TIME_FMT_ERROR	Error in input time format
PGSTD_E_TIME_VALUE_ERROR	Error in input time value
PGSTD_E_DATE_OUT_OF_RANGE	Input date is out of range of s/c clock
PGSTD_E_NO_LEAP_SECS	Leap seconds correction unavailable at requested time
PGS_E_TOOLKIT	An unexpected error occurred

EXAMPLES:

C:

```
char              asciiUTC[28];
PGSt_scTime       scTime[8];
PGSt_SMF_status   returnStatus;

strcpy(asciiUTC,”1995-02-04T12:23:44.125438Z”);

returnStatus = PGS_TD_UTC_to_SCtime(PGSd_EOS_AM,asciiUTC,
                                    scTime);
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
           :
           :
}
```

FORTRAN:

```
                  implicit none

                  integer           pgs_td_utc_to_sctime
                  character*27asciiutc
                  character*8       sctime
                  integer           returnstatus

                  asciiutc = ‘1995-02-04t12:23:44.125438Z’

                  returnstatus = pgs_td_utc_to_sctime(pgsd_eos_am,asciiutc,
                                                      sctime)
                  if (returnstatus .ne. pgs_s_success) then
                            :
c                 *** do some error handling ***

                      :

                  endif
```

NOTE: WARNING: To properly convert times to or from TRMM s/c clock time the value of the TRMM Universal Time Correlation Factor (UTCF) must be known. This value must be supplied by the user in the Process Control File (PCF). The following line MUST be contained in the PCF for any process that is converting to or from TRMM s/c clock time:

10123|TRMM UTCF value|<UTC VALUE>

Where the proper value of the UTCF should be substituted for <UTC VALUE>.

There is no corresponding problem for AM1 clock time, which is specified to have an accuracy of 100 microseconds.

UTC is:	Coordinated Universal Time

See Section 6.2.7.2 (ASCII Time Formats)

The output spacecraft times vary in format. The supported spacecraft times are in the following formats:

TRMM	CUC (platform specific variant of CCSDS Unsegmented time code(CUC) used)
EOS AM	CDS (platform specific variant of CCSDS day segmented time code (CDS) used)
EOS PM	CUC

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK- 1170

Convert Spacecraft Clock Time to UTC Time

NAME: **PGS_TD_SCtime_to_UTC( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_SCtime_to_UTC(
        PGSt_tag      spacecraftTag,
        PGSt_scTime   scTime[][8],
        PGSt_integer  numValues,
        char          asciiUTC[28],
        PGSt_double   offsets[])
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD.f'
include'PGS_TD_3.f'

integer function pgs_td_sctime_to_utc(spacecrafttag,
                                              sctime,numvalues,asciiutc,
                                              offsets)
        integer             spacecrafttag
        character*8         sctime(*)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
```

DESCRIPTION: This tool converts spacecraft clock time in platform dependent CCSDS format to UTC in CCSDS ASCII Time Code A format.

INPUTS: spacecraftTag-Spacecraft identifier, must be one of: PGSd_TRMM, PGSd_EOS_AM, PGSd_EOS_PM

scTime-Array of spacecraft clock times in platform dependent CCSDS format. UNITS, MAX, and MIN depend on the spacecraft, see the files containing the specific conversions for more information.

numValues-number of elements in the input scTime array (and therefore the output offsets array)

OUTPUTS:

Table 6-88. PGS_TD_SCtime_to_UTC Outputs

NAME	DESCRIPTION	UNITS
asciiUTC	UTC time of first s/c clock time in input array (in CCSDS ASCII Time Code A format). The values of MAX, and MIN depend on the spacecraft, add values from prologs!	ASCII
offsets	Array of offsets of each input s/c clock time in input array scTime relative to the first time in the array. This includes the first time as well (i.e., the first offset will be 0.0). The values of MAX, and MIN depend on the first time as well the spacecraft. Add values from prologs!	seconds

RETURNS:

Table 6-89. PGS_TD_SCtime_to_UTC Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSTD_W_BAD_SC_TIME	one or more input s/c times could not be deciphered
PGSTD_E_BAD_INITIAL_TIME	the initial input s/c time (first time in input array) could not be deciphered
PGSTD_E_SC_TAG_UNKNOWN	unknown/unsupported spacecraft ID tag
PGS_E_TOOLKIT	an unexpected error occurred

EXAMPLES:

C:

```
#define ARRAY_SIZE 1000

PGSt_scTime       scTime[ARRAY_SIZE][8];
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE];
PGSt_SMF_status   returnStatus;

*** Initialize scTime array ***
          :
          :

returnStatus = PGS_TD_SCtime_to_UTC(PGSd_EOS_AM,scTime,
                                    ARRAY_SIZE,asciiUTC,
                                    offsets);
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
          :
          :
}
```

FORTRAN:

```
implicit none

integer           pgs_td_sctime_to_utc
integer           array_size
character*8       sctime(array_size)
character*27asciiutc
```

```
double precision  offsets(array_size)
integer           returnstatus

*** Initialize sctime array ***
            :
            :
returnstatus = pgs_td_sctime_to_utc(pgsd_eos_am,sctime,
                                    array_size,asciiutc,
                                    offsets)
if (returnstatus .ne. pgs_s_success) then
            :
*** do some error handling ***
            :
endif
```

NOTES: WARNING: To properly convert times to or from TRMM s/c clock time the value of the TRMM Universal Time Correlation Factor (UTCF) must be known. This value must be supplied by the user in the Process Control File (PCF). The following line MUST be contained in the PCF for any process that is converting to or from TRMM s/c clock time:

10123|TRMM UTCF value|<UTC VALUE>

Where the proper value of the UTCF should be substituted for <UTC VALUE>.

There is no corresponding problem for AM1 clock time, which is specified to have an accuracy of 100 microseconds.

This function converts an array of input s/c times to an initial time and an array of offsets relative to this initial time. If the first time in the input array cannot be deciphered, this function returns an error. If any other time in the input array cannot be deciphered, the corresponding offset is set to PGSd_GEO_ERROR_VALUE and this function continues after setting the return value to a warning.

See Section 6.2.7.2 (ASCII Time Formats)

The input spacecraft times vary in format. The supported spacecraft times are in the following formats:

TRMM	CUC (platform specific variant of CUC used)
EOS AM	CDS (platform specific variant of CDS used)
EOS PM	CUC

UTC:	Coordinated Universal Time
TAI:	International Atomic Time
CUC:	CCSDS Unsegmented Time Code
CDS	CCSDS Day Segmented Time Code

REQUIREMENTS: PGSTK-1170

Convert CCSDS ASCII Time Format A to Format B

NAME: **PGS_TD_ASCIItime_AtoB( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_ASCIItime_AtoB(
        char    asciiUTC_A[28],
        char    asciiUTC_B[27]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_asciitime_atob(asciiutc_a,asciiutc_b);
        character*27   asciiutc_a
        character*26   asciiutc_b
```

DESCRIPTION: This Tool converts UTC time in CCSDS ASCII Time Code A to CCSDS ASCII Time Code B.

INPUTS:

Table 6-90. PGS_TD_ASCIItime_AtoB Inputs

Name	Description	Units	Min	Max
asciiUTC_A	UTC Time in CCSDS ASCII Time Code A	N/A	N/A	N/A

OUTPUTS:

Table 6-91. PGS_TD_ASCIItime_AtoB Outputs

Name	Description	Units	Min	Max
asciiUTC_B	UTC Time in CCSDS ASCII Time Code B	N/A	N/A	N/A

RETURNS:

Table 6-92. PGS_TD_ASCIItime_AtoB Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_TIME_VALUE_ERROR	Error in input time value
PGSTD_E_TIME_FMT_ERROR	Error in input time format
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnValue;
char              asciiUTC_A[28];
char              asciiUTC_B[27];

strcpy(asciiUTC_A,”1998-06-30T10:51:28.320000Z”);
returnValue = PGS_TD_ASCIItime_AtoB(asciiUTC_A,asciiUTC_B);
if (returnValue != PGS_S_SUCCESS)
{
** test errors, take appropriate action **
   :
   :
}
printf(“%s\n”,asciiUTC_B);
```

FORTRAN:

```
implicit none

integer           pgs_td_asciitime_atob
integer           returnvalue
character*27asciiutc_a
character*26asciiutc_b

asciiutc_a = ‘1998-06-30T10:51:28.320000’
returnvalue = pgs_td_asciitime_atob(asciiutc_a,asciiutc_b)
if (returnvalue .ne. pgs_s_success) goto 999
write(6,*) asciiutc_b
```

NOTES: The output of this tool is in CCSDS ASCII Time Code B format.

See Section 6.2.7.2 (ASCII Time Formats)

REQUIREMENTS: PGSTK-1170, PGSTK-1180, PGSTK-1210

Convert CCSDS ASCII Time Format B to Format A

NAME: **PGS_TD_ASCIItime_BtoA( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_ASCIItime_BtoA(
        char    asciiUTC_B[27],
        char    asciiUTC_A[28]);
```

FORTRAN:

```
include‘PGS_SMF.f’
include‘PGS_TD_3.f’

integer function pgs_td_asciitime_btoa(asciiutc_b,asciiutc_a);
        character*26   asciiutc_b
        character*27   asciiutc_a
```

DESCRIPTION: This Tool converts UTC time in CCSDS ASCII Time Code B to CCSDS ASCII Time Code A.

INPUTS:

Table 6-93. PGS_TD_ASCIItime_BtoA Inputs

Name	Description	Units	Min	Max
asciiUTC_B	UTC Time in CCSDS ASCII Time Code B	N/A	N/A	N/A

OUTPUTS:

Table 6-94. PGS_TD_ASCIItime_BtoA Outputs

Name	Description	Units	Min	Max
asciiUTC_A	UTC Time in CCSDS ASCII Time Code A	N/A	N/A	N/A

RETURNS:

Table 6-95. PGS_TD_ASCIItime_BtoA Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_TIME_VALUE_ERROR	Error in input time value
PGSTD_E_TIME_FMT_ERROR	Error in input time format
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnValue;
char              asciiUTC_B[27];
char              asciiUTC_A[28];

strcpy(asciiUTC_B,”1998-181T10:51:28.320000Z”);
returnValue = PGS_TD_ASCIItime_BtoA(asciiUTC_B,asciiUTC_A);
if (returnValue != PGS_S_SUCCESS)
{
** test errors, take appropriate action **
      :
      :
}
printf(“%s\n”,asciiUTC_A);
```

FORTRAN:

```
implicit none

integer           pgs_td_asciitime_btoa
integer           returnvalue
character*26asciiutc_b
character*27asciiutc_a

asciiutc_b = ‘1998-181T10:51:28.320000’
returnvalue = pgs_td_asciitime_btoa(asciiutc_b,asciiutc_a)
if (returnvalue .ne. pgs_s_success) goto 999
write(6,*) asciiutc_a
```

NOTES: The output of this tool is in CCSDS ASCII Time Code A format.

See Section 6.2.7.2 (ASCII Time Formats)

REQUIREMENTS: PGSTK-1170, PGSTK-1180, PGSTK-1210

Convert UTC to GPS Time

NAME: **PGS_TD_UTCtoGPS( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoGPS(
        char            asciiUTC[28],
        PGSt_double     *secGPS);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_utctogps(asciiUTC,secgps)
        character*27            asciiutc
        double precision        secgps
```

DESCRIPTION: This tool converts from UTC time to GPS time.

INPUTS:

Table 6-96. PGS_TD_UTCtoGPS Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A or B format	time	1961-01-01 T00:00:00	2008-03-30 T23:59:59.999999

OUTPUTS:

Table 6-97. PGS_TD_UTCtoGPS Outputs

Name	Description	Units	Min	Max
secGPS	Continuous real seconds since 0 hrs UTC on Jan. 6, 1980	seconds	-599961636.577182	890956802.999999

RETURNS:

Table 6-98. PGS_TD_UTCtoGPS Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available input time
PGSTD_E_TIME_FMT_ERROR	Error in format of ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of the ASCII UTC time
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
char             asciiUTC[28];
PGSt_double      secGPS;
PGSt_SMF_status  returnStatus;
char             err[PGS_SMF_MAX_MNEMONIC SIZE]
char             msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus = PGS_TD_UTCtoGPS(asciiUTC,&secGPS);
if(returnStatus != PGS_S_SUCCESS)
{
   PGS_SMF_GetMsg(&returnStatus, err, msg);
   printf(“\n ERROR:  %s”, msg);
}
```

FORTRAN:

```
implicit none

integer          pgs_td_utctogps
character*27asciiutc
double precision  secgps
integer          returnstatus
integer          anerror
character*35errname
character*150    errmsg

returnstatus = pgs_td_utctogps(asciiutc,secgps)
if(returnstatus .ne. PGS_S_SUCCESS) then
   returnstatus = pgs_smf_getmsg(anerror,errorname,errmsg)
   write(*,*) errname,errmsg
endif
```

NOTES: See Section 6.2.3.2 (ASCII Time Formats)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

GPS:	Global Positioning System
TAI:	International Atomic Time
UTC:	Coordinated Universal Time

REQUIREMENTS: PGSTK-1170, PGSTK-1210

Convert GPS to UTC Time

NAME: **PGS_TD_GPStoUTC( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_Status
PGS_TD_GPStoUTC(
        PGSt_double   secGPS,
        char          asciiUTC[28]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_gpstoutc(secgps, asciiutc)
        double precision        secgps
        character*27            asciiutc
```

DESCRIPTION: This tool converts from GPS time to UTC time.

INPUTS:

Table 6-99. PGS_TD_GPStoUTC Inputs

Name	Description	Units	Min	Max
secGPS	Continuous real seconds since 0 hrs UTC on Jan. 6, 1980	seconds	-599961636.577182	see NOTES

OUTPUTS:

Table 6-100. PGS_TD_GPStoUTC Outputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A	time	1961-01-01	see NOTES

RETURNS:

Table 6-101. PGS_TD_GPStoUTC Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap seconds correction for input time
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
char              asciiUTC[28];
PGSt_double       secGPS;
PGSt_SMF_status   returnStatus;
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus = PGS_TD_GPStoUTC(secGPS,asciiUTC);
if(returnStatus != PGS_S_SUCCESS)
{
   PGS_SMF_GetMsg(&returnStatus, err, msg);
   printf(“\n ERROR:  %s”, msg);
}
```

FORTRAN:

```
implicit none

integer           pgs_td_gpstoutc
character*27asciiutc
double precision  secgps
integer           returnstatus
integer           anerror
character*35errname
character*150     errmsg

returnstatus =  pgs_td_gpstoutc(secgps,asciiUTC)
if(returnstatus .ne. PGS_S_SUCCESS) then
   returnstatus = pgs_smf_getmsg(anerror,errorname,errmsg)
   write(*,*) errname,errmsg
endif
```

NOTES: See Section 6.2.3.2 (ASCII Time Formats)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

GPS:	Global Positioning System
TAI:	International Atomic Time
UTC:	Coordinated Universal Time

REQUIREMENTS: PGSTK-1170, PGSTK-1210

Convert UTC Time to TDT Time

NAME: **PGS_TD_UTCtoTDTjed( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoTDTjed(
        char            asciiUTC[28],
        PGSt_double     jedTDT[2]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_utctotdtjed(asciiutc, jedtdt)
        character*27            asciiutc
        double precision        jedtdt(2)
```

DESCRIPTION: This tool converts UTC in CCSDS ASCII time format A or B to TDT as a Julian date (TDT = Terrestrial Dynamical Time)

INPUTS:

Table 6-102. PGS_TD_UTCtoTDTjed Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII time Code A or B format	time	1961-01-01	see NOTES

OUTPUTS:

Table 6-103. PGS_TD_UTCtoTDTjed Outputs

Name	Description	Units	Min	Max
jedTDT	TDT as a Julian date	days	see NOTES	see NOTES

RETURNS:

Table 6-104. PGS_TD_UTCtoTDTjed Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_TIME_FMT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGSTD_E_NO_LEAP_SECS	Leap second errors
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28] =
                  “2002-06-30T11:04:57.987654Z”;
PGSt_double       jedTDT[2];
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus=PGS_TD_UTCtoTDTjed(asciiUTC,jedTDT);
if (returnStatus != PGS_S_SUCCESS)
     {
           PGS_SMF_GetMsg(&returnStatus,err,msg);
           printf(“\nERROR: %s”,msg)
     }
```

FORTRAN:

```
implicit none

integer           pgs_td_utctotdtjed
integer           returnstatus
character*27asciiutc
double precision  jedtdt(2)
character*33      err
character*241     msg

asciiutc = ‘1998-06-30T10:51:28.320000Z’
returnstatus = pgs_td_utctotdtjed(asciiutc,jedtdt)
if (returnstatus .ne. pgs_s_success)
         returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
         write(*,*) err, msg
endif
```

NOTES: **TIME ACRONYMS:**

TDT is:	Terrestrial Dynamical Time
UTC is:	Coordinated Universal Time

Prior to 1984, there is no distinction between TDT and TDB; either one is denoted “ephemeris time” (ET). Also, the values before 1972 are based on U.S. Naval Observatory estimates, which are the same as adopted by the JPL Ephemeris group that produces the DE series of solar system ephemerides, such as DE200.

Section 6.2.7.4 (Toolkit Julian Dates)

See Section 6.2.7.2 (ASCII Time Formats)

See See Section 6.2.7.5.1 (TAI-UTC Boundaries)

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK-1215

Convert UTC Time to TDB Time

NAME: **PGS_TD_UTCtoTDBjed( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoTDBjed(
        char            asciiUTC[28],
        PGSt_double     jedTDB[2]);
```

FORTRAN:

```
include‘PGS_SMF.f’
include‘PGS_TD_3.f’

integer function pgs_td_utctotdbjed(asciiutc, jedtdb)
        character*27            asciiutc
        double precision        jedtdb(2)
```

DESCRIPTION: This tool converts UTC in CCSDS ASCII time format A or B to TDB as a Julian date (TDB = Barycentric Dynamical Time)

INPUTS:

Table 6-105. PGS_TD_UTCtoTDBjed Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII time Code A or B format	time	1961-01-01	see NOTES

OUTPUTS:

Table 6-106. PGS_TD_UTCtoTDBjed Outputs

Name	Description	Units	Min	Max
jedTDB	TDB as a Julian date	days	see NOTES	see NOTES

RETURNS:

Table 6-107. PGS_TD_UTCtoTDBjed Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_TIME_FMT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGSTD_E_NO_LEAP_SECS	Leap second errors
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28] =
                  “2002-02-23T11:04:57.987654Z”;
PGSt_double       jedTDB[2];
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus=PGS_TD_UTCtoTDBjed(asciiUTC,jedTDB);
if (returnStatus != PGS_S_SUCCESS)
     {
         PGS_SMF_GetMsg(&returnStatus,err,msg);
         printf(“\nERROR: %s”,msg)
     }
```

FORTRAN:

```
implicit none

integer           pgs_td_utctotdbjed
integer           returnstatus
character*27asciiutc
double precision  jedtdb(2)
character*33err
character*241     msg

asciiutc = ‘1998-06-30T10:51:28.320000Z’
returnstatus = pgs_td_utctotdbjed(asciiutc,jedtdb)
if (returnstatus .ne. pgs_td_utctotdbjed(asciiutc,jedtdb)
         returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
         write(*,*) err, msg
endif
```

NOTES: **TIME ACRONYMS:**

TDB is: Barycentric Dynamical Time
UTC is: Coordinated Universal Time

Prior to 1984, there is no distinction between TDT and TDB; either one is denoted “ephemeris time” (ET). Also, the values before 1972 are based on U.S. Naval Observatory estimates, which are the same as adopted by the JPL Ephemeris group that produces the DE series of solar system ephemerides, such as DE200.

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.4 (Toolkit Julian Dates)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK-1215

Compute Elapsed TAI Time

NAME: **PGS_TD_TimeInterval( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

pgs_status
PGS_TD_TimeInterval(
        PGSt_double   startTAI,
        PGSt_double   stopTAI,
        PGSt_double   *interval)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer function pgs_td_timeinterval( starttai, stoptai, interval)
        double precision        starttai
        double precision        stoptai
        double precision        interval
```

DESCRIPTION: This function computes the elapsed TAI time in seconds between any two time intervals

INPUTS:

Table 6-108. PGS_TD_TimeInterval Inputs

Name	Description	Units	Min	Max
startTAI	start time in TAI	seconds	none	none
stopTAI	stop time in TAI	seconds	none	none

OUTPUTS:

Table 6-109. PGS_TD_TimeInterval Outputs

Name	Description	Units	Min	Max
interval	elapsed time interval	seconds	none	none

RETURNS:

Table 6-110. PGS_TD_TimeInterval Returns

Return	Description
PGS_S_SUCCESS	Successful return

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       startTAI;
PGSt_double       stopTAI;
PGSt_double       interval;

startTAI = 34523.5;
stopTAI = 67543.2;
returnStatus = PGS_TD_TimeInterval(startTAI,stopTAI,
                                   &interval);
```

FORTRAN:

```
implicit none

integer           pgs_td_timeinterval
integer           returnstatus
double precision  starttai
double precision  stoptai
double precision  interval

returnstatus = pgs_td_timeinterval(starttai,stoptai,
                                   interval)
```

NOTES: This interval is the same as elapsed internal time and is the true interval in System International (SI) seconds.

REQUIREMENTS: PGSTK-1190

Convert UTC in CCSDS ASCII Format to Julian Date Format

NAME: **PGS_TD_UTCtoUTCjd()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoUTCjd(
        char            asciiUTC[28],
        PGSt_double     jdUTC[2])
```

FORTRAN:

```
include ‘PGS_SMF.f’
include ‘PGS_TD_3.f’

integer function pgs_td_utctoutcjd(asciiutc, jdutc)
character*27            asciiutc
double precision        jdutc(2)
```

DESCRIPTION: Converts ASCII UTC times to UTC Julian Dates

INPUTS:

Table 6-111. PGS_TD_UTCtoUTCjd Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII time Code A or B format	time	1961-01-01	see NOTES

OUTPUTS:

Table 6-112. PGS_TD_UTCtoUTCjd Outputs

Name	Description	Units	Min	Max
jdUTC[2]	UTC Julian date	days	none	none

RETURNS:

Table 6-113. PGS_TD_UTCtoUTCjd Returns

Return	Description
PGS_S_SUCCESS	successful return
PGSTD_M_LEAP_SEC_IGNORED	leap second portion of input time discarded
PGSTD_E_TIME_FMT_ERROR	error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	error in format of input ASCII UTC time
PGS_E_TOOLKIT	something unexpected happened, execution aborted

NOTES: Caution should be used because UTC Julian Date jumps backwards each time a leap second is introduced. Therefore, in a leap second interval the output times will repeat those in the previous second (provided that the UTC ASCII seconds field ran from 60.0 to 60.9999999 etc. as they should during that one second). Therefore, the only known uses for this function are:
(a) to get UT1, (after conversion to modified Julian Date by subtracting 2400000.5) by accessing an appropriate table of differences
(b) to determine the correct Julian Day at which to access any table based on UTC and listed in Julian date, such as leap seconds, UT1, and polar motion tables.

UTC is: Coordinated Universal Time

See section 6.2.7.4 (Toolkit Julian Dates)

REQUIREMENTS: PGSTK - 1170, 1220

Convert UTC Julian Date to CCSDS ASCII Time Code A Format

NAME: **PGS_TD_UTCjdtoUTC()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCjdtoUTC(
        PGSt_double   jdUTC[2],
        PGSt_boolean  onLeap,
        char          asciiUTC[28])
```

FORTRAN:

```
include ‘PGS_SMF.f’
include ‘PGS_TD_3.f’

integer function pgs_td_utcjdtoutc(jdutc,onleap,asciiutc)
double precision    jdutc(2)
integer             onleap
character*27        asciiutc
```

DESCRIPTION: This tool converts UTC as a Julian date to UTC in CCSDS ASCII Time Code A format.

INPUTS:

Table 6-114. PGS_TD_UTCjdtoUTC Inputs

Name	Description	Units
jdUTC	UTC time as a Julian date	days
onLeap	Indicates if input time is occurring during a leap second	T/F

OUTPUTS:

Table 6-115. PGS_TD_UTCjdtoUTC Outputs

Name	Description	Units
asciiUTC	UTC time in CCSDS ASCII Time Code A format	time

RETURNS:

Table 6-116. PGS_TD_UTCjdtoUTC Returns

Return	Description
PGS_S_SUCCESS	successful return
PGSTD_E_TIME_FMT_ERROR	a leap second was indicated at an inappropriate time
PGS_E_TOOLKIT	something unexpected happened

EXAMPLES:

C:

```
PGSt_SMF_status  returnStatus;

PGSt_double      jdUTC[2]={2449534.5,0.5};

char             asciiUTC[28];

returnStatus = PGS_TD_UTCjdtoUTC(jdUTC,PGS_FALSE,asciiUTC);

if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
            :
            :
}

/* asciiUTC now contains the value:
   “1994-07-01T12:00:00.000000Z” */

printf(“UTC: %s\n”,asciiUTC);
```

FORTRAN:

```
integer          pgs_td_utcjdtoutc

integer          returnstatus

double precision jdutc(2)

character*27     asciiutc


jdutc(1) = 2449534.5D0

jdutc(1) = 0.5D0

returnstatus = pgs_td_utcjdtoutc(jdutc,pgs_false,asciiutc)

if (returnstatus .ne. pgs_s_success) goto 999

!  asciiutc now contains the value:
! ‘1994-07-01T12:00:00.000000Z’

write(6,*) ‘UTC: ‘, asciiutc
```

NOTES: UTC is: Coordinated Universal Time

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems)

Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

See section 6.2.7.4 (Toolkit Julian Dates)

REQUIREMENTS: PGSTK - 1210, 1220, 1160, 1170

Convert UTC to UT1

NAME: **PGS_TD_UTCtoUT1( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoUT1(
        char            asciiUTC[28],
        PGSt_double     *secUT1);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'
include'PGS_CSC_4.f'

integer function pgs_td_utctout1(asciiutc, secut1)
        character*27            asciiutc
        double precision        secut1
```

DESCRIPTION: This tool converts a time from CCSDS ASCII Time (Format A or B) to UT1

INPUTS:

Table 6-117. PGS_TD_UTCtoUT1 Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A or B format	time	1979-06-30T00:00:00 also see notes	Date

OUTPUTS:

Table 6-118. PGS_TD_UTCtoUT1 Outputs

Name	Description	Units	Min	Max
secUT1	UT1 in seconds from midnight	sec	0.0	86400.999999

RETURNS:

PGS_S_SUCCESS
PGSTD_E_TIME_FMT_ERROR
PGSTD_E_TIME_VALUE_ERROR
PGSCSC_W_PREDICTED_UT1
PGSTD_E_NO_UT1_VALUE
PGS_E_TOOLKIT

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus
char        asciiUTC[28] = “2002-07-27T11:04:57.987654Z
PGSt_double       secUT1
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus=PGS_TD_UTCtoUT1(asciiUTC,&secUT1);
if  (returnStatus != PGS_S_SUCCESS)
{
   PGS_SMF_GetMsg(&returnStatus,err,msg);
   printf(“\nERROR: %s”,msg)
}
```

FORTRAN:

```
implicit none

integer           pgs_td_utctout1
integer           returnstatus
character*27asciiutc
double precision  secut1
character*33      err
character*241     msg

asciiutc = ‘2002-07-27T11:04:57.987654Z’
returnstatus = pgs_td_utctout1(asciiutc,secut1)
if (returnstatus .ne. pgs_s_success) then
   returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
   write(*,*) err, msg
endif
```

NOTES: Although UT1 was used for civil timekeeping before Jan. 1, 1972, today UT1 is a measure of Earth rotation only; it is a measure of the angle of the Greenwich Meridian from the equinox of date such that 24 hours of System International (SI) seconds (86400 seconds) of TAI or TDT constitute one full revolution. As such, it can be directly reduced to Greenwich Apparent Sidereal Time (GAST). This function should be used with caution near midnight. For example, if UTC is 0.5 seconds before midnight, and UT1 - UTC = 0.6 s, then this function returns 0.1 s, but the day has changed.

Prior to Jan. 1, 1972, either UT1 or, for a brief period, a variant called UT2 that accounts for some of the periodic nonuniformities of Earth rotation, were used for time keeping.

TIME ACRONYMS:

UT1 is:	Universal Time
UTC is:	Coordinated Universal Time

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems), Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK-1215

Convert UTC to UT1 Julian Date

NAME: **PGS_TD_UTCtoUT1jd( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoUT1jd(
          char            asciiUTC[28],
          PGSt_double     jdUT1[2])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'

integer function pgs_td_utctout1jd(asciiutc, jdut1)
          character*27            asciiutc
          double precision        jdut1(2)
```

DESCRIPTION: This tool converts a time from CCSDS ASCII Time (Format A or B) to Julian date.

INPUTS:

Table 6-119. PGS_TD_UTCtoUT1jd Inputs

Name	Description	Units
asciiUTC	UTC time in CCSDS ASCII Time Code A format or ASCII Time Code B format	ASCII

OUTPUTS:

Table 6-120. PGS_TD_UTCtoUT1jd Outputs

Name	Description	Units
jdUT1	UT1 Julian date as two real numbers, the first a half integer number of days and the second the fraction of a day between this half integer number of days and the next half integer day number.	days

RETURNS:

Table 6-121. PGS_TD_UTCtoUT1jd Returns

Return	Description
PGS_S_SUCCESS	Successful execution
PGSTD_M_LEAP_SEC_IGNORED	Leap second portion of input time discarded
PGSTD_E_TIME_FMT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGS_E_TOOLKIT	Something unexpected happened, execution aborted

EXAMPLES: None

NOTES: Although UT1 was used for civil timekeeping before Jan. 1, 1972, today UT1 is a measure of Earth rotation only; it is a measure of the angle of the Greenwich Meridian from the equinox of date such that 24 hours of System International (SI) seconds (86400 seconds) of TAI or TDT constitute one full revolution. As such, it can be directly reduced to Greenwich Apparent Sidereal Time (GAST).

Prior to Jan. 1, 1972, either UT1 or, for a brief period, a variant called UT2 that accounts for some of the periodic nonuniformities of Earth rotation, were used for time keeping.

TIME ACRONYMS:

UT1 is:	Universal Time
UTC is:	Coordinated Universal Time

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.4 (Toolkit Julian Dates)

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK-1170, PGSTK-1210

Get Leap Second

NAME: **PGS_TD_LeapSec()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>
PGSt_SMF_status
PGS_TD_LeapSec(
        PGSt_double  jdUTC[2],
        PGSt_double  *leapSec,
        PGSt_double  *lastChangeJD,
        PGSt_double  *nextChangeJD,
        char         *leapStatus)
```

FORTRAN

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer funtion pgs_td_leapsec(jdutc,leapsec,lastchangejd,nextchangejd,
                               leapstatus
double precision     jdutc(2)
double precision     leapsec
double precision     lastchangejd
double precision     nextchangejd
character*10         leapstatus
```

DESCRIPTION: This tool accesses the file 'leapsec.dat', extracts the leap second value for an input Julian Day number, and returns an error status.

INPUTS:

Table 6-122. Get Leap Second Inputs

Name	Description	Units	Min	Max
jdUTC	UTC Julian Day number	days (see NOTES)	N/A	N/A

OUTPUTS:

Table 6-123. Get Leap Second Outputs

Name	Description	Units	Min	Max
leapSec	leap second value for day jdUTC, read from table	seconds	0	N/A
lastChangeJD	Julian Day number upon which that leap second value was effective	days (see NOTES)	N/A	N/A
nextChangeJD	Julian Day number of the next ACTUALor PREDICTED leap second	days (see NOTES)	N/A	N/A
leapStatus	indicates whether the leap second value is ACTUAL, PREDICTED, a LINEARFIT, or ZEROLEAPS (leap second value is set to zero if the input time is before the start of the table)	N/A	N/A	N/A

RETURNS:

Table 6-124. Get Leap Seconds Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSTD_W_JD_OUT_OF_RANGE	invalid input Julian Day number
PGSTD_W_DATA_FILE_MISSING	leap second file not found

EXAMPLES:

```
PGSt_double        jdUTC[2];
PGSt_double        leapsecond;
PGSt_double        lastChangeJD;
PGSt_double        nextChangeJD;

PGSt_SMF_status    returnStatus;
char           leapStatus[10];

jdUTC[0] = 2439999.5;
jdUTC[1] = 0.5;
returnStatus = PGS_TD_LeapSec(jdUTC,&leapsecond,
                             &lastChangeJD,
                             &nextChangeJD,leapStatus);
if (returnStatus != PGS_S_SUCCESS)
{
/*  handle errors  */
}
```

NOTES:

With Toolkit 5.2, the functions that call PGS_TD_LeapSec() will return an error and write a diagnostic message to the Log Status File indicating that an obsolete format was encountered in the Leap Seconds file, if they encounter the "PREDICTED" status.

UTC: Coordinated Universal Time

TAI: International Atomic Time

REQUIREMENTS: PGSTK - 1050, 0930

6.2.7.8 TD Functions

PGS_TD_ADEOSIItoTAI

This tool converts ADEOS-II s/c clock time (instrument time + pulse time) to TAI (prototype code).

PGS_TD_ADEOSIItoUTC

This tool converts converts ADEOS-II s/c clock time (instrument time + pulse time) to a UTC string in CCSDS ASCII Time Code A format (prototype code).

PGS_TD_ASCIItime_AtoB

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_ASCIItime_BtoA

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_EOSAMtoTAI

This function converts EOS AM spacecraft clock time in CCSDS day segmented Time Code (CDS) (with implicit P-field) format to TAI (as real continuous seconds since 12AM UTC 1-1-1993).

PGS_TD_EOSAMtoUTC

This function converts EOS AM spacecraft clock time in platform-dependent format to UTC in CCSDS ASCII time code A format.

PGS_TD_EOSPMtoTAI

This function converts EOS PM spacecraft clock time in CCSDS Unsegmented Time Code (CUC) (with explicit P-field) format to TAI (as real continuous seconds since 12AM UTC 1-1-1993).

PGS_TD_EOSPMtoUTC

This function converts EOS PM spacecraft clock time in CCSDS unsegmented Time Code (CUC) (with explicit P-field) format to UTC in CCSDS ASCII time code A format.

PGS_TD_FGDCtoUTC

This function converts an FGDC ASCII date string and time string to CCSDS ASCII Time Code (format A). The input FGDC time string may be in “Universal Time” or “local time” format.

PGS_TD_GPStoUTC

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_ISOinttoTAI

This function converts an integer number that represents an ISO time (YYMMDDhh) to TAI.

PGS_TD_ISOinttoUTCjd

This function converts an integer number that represents an ISO time (YYMMDDhh) to a UTC time in toolkit Julian date format.

PGS_TD_JDtoMJD

This function converts a Julian date to a modified Julian date.

PGS_TD_JDtoTJD

This function converts a Julian date to a truncated Julian date.

PGS_TD_JulianDateSplit

This function converts a Julian date to Toolkit Julian date format

PGS_TD_LeapSec

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_MJDtoJD

This function converts a modified Julian date to a Julian date.

PGS_TD_PB5CtoUTCjd

This function converts a time in PB5C time format to TAI (Toolkit internal time).

PGS_TD_PB5toTAI

This function converts a time in PB5 time format to TAI (Toolkit internal time).

PGS_TD_PB5toUTCjd

This function converts a time in PB5 time format to UTC time in toolkit Julian date format.

PGS_TD_SCtime_to_UTC

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAIjdtoTAI

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAIjdtoTDTjed

This function converts TAI Julian date to TDT Julian ephemeris date.

PGS_TD_TAIjdtoUTCjd

This function converts TAI Julian date to UTC Julian date.

PGS_TD_TAItoGAST

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAItoISOint

This function converts TAI to an integer number that represents an ISO time (YYMMDDhh).

PGS_TD_TAItoTAIjd

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAItoUDTF

This tool converts TAI to a UDTF integer array.

PGS_TD_TAItoUT1jd

This tool converts continuous seconds since 12AM UTC 1-1-93 to UT1 time as a Julian date.

PGS_TD_TAItoUT1pole

This tool converts continuous seconds since 12AM UTC 1-1-93 to UT1 time as a Julian date and returns x and y polar wander values and UT1-UTC as well.

PGS_TD_TAItoUTC

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAItoUTCjd

This tool converts continuous seconds since 12AM UTC 1-1-93 to UTC time as a Julian date.

PGS_TD_TDBjedtoTDTjed

This function converts TDB (Barycentric Dynamical Time) as a Julian ephemeris date to TDT (Terrestrial Dynamical Time) as a Julian ephemeris date.

PGS_TD_TDTjedtoTAIjd

This function converts TDT Julian ephemeris date to TAI Julian date.

PGS_TD_TDTjedtoTDBjed

This function converts TDT (Terrestrial Dynamical Time) as a Julian ephemeris date to TDB (Barycentric Dynamical Time) as a Julian ephemeris date.

PGS_TD_TJDtoJD

This function converts a truncated Julian date to a Julian date.

PGS_TD_TRMMtoTAI

This function converts TRMM spacecraft clock time in CCSDS Unsegmented Time Code (CUC) (with implicit P-field) format to TAI (Toolkit internal time).

PGS_TD_TRMMtoUTC

This function converts TRMM spacecraft clock time in CCSDS unsegmented Time Code (CUC) (with implicit P-field) format to UTC in CCSDS ASCII time code A format.

PGS_TD_TimeInterval

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UDTFtoTAI

This function converts a UDTF integer array to TAI.

PGS_TD_UDTFtoUTCjd

This function converts a UDTF integer array to a UTC Julian date.

PGS_TD_UT1jdtoUTCjd

This tool converts UT1 time as a Julian date to UTC time as a Julian date.

PGS_TD_UTC_to_SCtime

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCjdtoISOint

This function converts a UTC time in toolkit Julian date format to an integer number that represents an ISO time (YYMMDDhh).

PGS_TD_UTCjdtoPB5

This function converts a UTC time in toolkit Julian date format to PB5 time format.

PGS_TD_UTCjdtoPB5C

This function converts a UTC time in toolkit Julian date format to PB5C time format.

PGS_TD_UTCjdtoTAIjd

This tool converts UTC as a Julian date to TAI as a Julian date.

PGS_TD_UTCjdtoUT1jd

This tool converts UTC time as a Julian date to UT1 time as a Julian date.

PGS_TD_UTCjdtoUTC()

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoADEOSII

This function converts UTC in CCSDS ASCII time code A (or B) format to ADEOS s/c clock format (this is a prototype only).

PGS_TD_UTCtoEOSAM

This function converts UTC in CCSDS ASCII time code A (or B) format to EOS AM spacecraft (s/c) clock time in CCSDS Day Segmented (CDS) Time Code (with implicit P-field) format.

PGS_TD_UTCtoEOSPM

This function converts UTC in CCSDS ASCII Time Code A or CCSDS ASCII Time Code B format to EOS PM spacecraft clock time in CCSDS Unsegmented Time Code (CUC) (with explicit P-field) format.

PGS_TD_UTCtoFGDC

This function converts UTC Time in CCSDS ASCII Time Code (format A or B) to the equivalent FGDC ASCII date string and time string. The time string will be in “Universal Time” or “local time” format depending on the value of the input variable tdf.

PGS_TD_UTCtoGPS

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTAI

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTAIjd

This tool converts UTC in CCSDS ASCII time format A or B to TAI as a Julian date.

PGS_TD_UTCtoTDBjed

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTDTjed

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTRMM()

This function converts UTC in CCSDS ASCII time code A (or B) format to TRMM spacecraft (s/c) clock time in CCSDS Unsegmented Time Code (CUC) (with implicit P-field) format.

PGS_TD_UTCtoUT1

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoUT1jd

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoUTCjd

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_calday

This function converts Julian day to calendar day (year, month, day).

PGS_TD_gast

This function converts GMST, nutation in longitude and TDB Julian date to Greenwich Apparent Sidereal Time expressed as the hour angle of the true vernal equinox of date at the Greenwich meridian (in radians).

PGS_TD_gmst

The function converts UT1 expressed as a Julian day to Greenwich Mean Sidereal Time, i.e. the hour angle of the vernal equinox at the Greenwich meridian (in radians).

PGS_TD_julday

This function converts calendar day (year, month, dat) to Julian day.

PGS_TD_sortArrayIndices

This function sorts an array of PGSt_double (double precision) numbers in ascending order.

PGS_TD_timeCheck

This function accepts a character array (string) as an input and returns a value indicating if the string is in a valid CCSDS ASCII format.